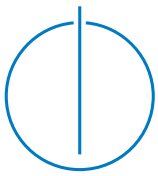
FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Computer Science

Spatial and Temporal Interpolation of Multi-View Video

Tobias Gurdan



FAKULTÄT FÜR INFORMATIK

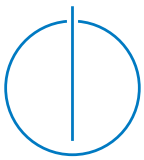
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Computer Science

Spatial and Temporal Interpolation of Multi-View Video

Räumliche und Zeitliche Interpolation von Videoaufnahmen mehrerer Ansichten

Author:	Tobias Gurdan
Supervisor:	Prof. Dr. Daniel Cremers
Advisors:	Dipl.-Inf. Martin Oswald,
	Dipl.-Ing. Daniel Gurdan
Date:	November 14, 2013



ASCENDING TECHNOLOGIES

I assure the single handed composition of this bachelor's thesis only supported by declared resources.

Munich, November 14, 2013 Tobias Gurdan

Abstract

Image interpolation is one of the vital tools in image processing for the creation of visual effects. This thesis proposes a framework for multi-view video interpolation in space and time. It aims to develop methods that impose as few constraints on setup and input as possible, allowing for low-cost and intuitive usage. Algorithms for both sparse and dense image matching and warping are being presented, evaluated and further enhanced. Additionally, the implementation and applied optimization techniques are outlined. After presenting applications and results, the thesis closes discussing the capabilities and limitations of the proposed algorithms.

Zusammenfassung

Die Interpolation von Bildern ist eine der vorrangigen Methoden in der Bildbearbeitung, um ansprechende visuelle Effekte zu erzeugen. Diese Arbeit stellt ein Framework für räumliche und zeitliche Interpolation von Videos mehrerer Ansichten vor. Die dabei entwickelten Methoden sollen geringe Anforderungen an Kameras, Szenen, Aufbau und Benutzereingaben stellen und somit eine kostengünstige und einfach zu bedienende Lösung bieten. Algorithmen sowohl für dünn besetztes als auch dichtes matchen und deformieren von Bildern werden vorgestellt, bewertet und weiterentwickelt. Zudem werden die erarbeitete Implementierung sowie verwendete Optimierungstechniken umrissen. Der letzte Teil der Arbeit präsentiert Ergebnisse und mögliche Anwendungsszenarien. Die Arbeit schließt mit einer Bewertung der Möglichkeiten und Limitierungen der vorgestellten Methoden.

Acknowledgements

I would like to express my deep gratitude to my advisors, Martin Oswald and Daniel Gurdan, who greatly supported me over the last few months and always had an open ear and mind. I would also like to thank Prof. Dr. Cremers for making this thesis possible in the first place.

A very special thanks goes to my brother, with which I was able to share an amazing time during the course of this work. He always encouraged me, accompanied me through most of the ups and downs and was and always will be an inspiring example.

My never-ending gratitude also goes to the rest of my loving family, Cornelia, Johann, Manuela, Christian and Silvia, who believed in me from the beginning until today. They always have an open door and heart, and provide a shelter when I need it the most.

Furthermore, I want to thank all of my friends that never got tired of me and my unbreakable spirit, and share with me the same enjoyment of life; Lorenz Bergler, Katharina Hartwig, Sebastian Kalchgruber, Michael Fischer, Christoph Rauch, Markus Kaiser. I especially want to acknowledge the latter two as well as my advisors for their elaborate revisions and great help in the final making of this document. In addition, Markus shared an office with me and I want to thank him again for the great time we had together.

Finally, this thesis would not have been possible without the cooperation with Ascending Technologies GmbH. My heartfelt thanks to the company and all its employees, who made the time I spent on this work even more enjoyable.

Danksagungen

Zunächst möchte ich mich ganz herzlich bei meinen Betreuern, Martin Oswald und Daniel Gurdan, bedanken, die mich über die letzten Monate hinweg großartig unterstüzt haben, stets ein offenes Ohr hatten und für alle Ideen und Vorschläge empfänglich waren. Ich möchte auch Herrn Prof. Dr. Cremers danken, durch dessen Einverständnis und Vermittlung diese Arbeit überhaupt erst zu Stande kommen konnte.

Meine äußerste Dankbarkeit gilt meinem Bruder, mit dem ich im Laufe dieser Arbeit eine unvergessliche Zeit erleben durfte. Er spornte mich stets an, begleitete mich durch viele Höhen und Tiefen und er war und wird immer ein inspirierendes Vorbild für mich sein.

Meine nicht endende Dankbarkeit geht auch an den Rest meiner Familie, Cornelia, Johann, Manuela, Christian und Silvia, die vom Anbeginn bis heute an mich glaubten. Sie haben stets eine offene Tür und ein offenes Herz und bieten Zuflucht, wenn ich sie brauche.

Weiterhin will ich meinen Freunden danken, die mir und meinem standhaften Willen nie müde wurden, wir teilen die gleiche Freude am Leben; Lorenz Bergler, Katharina Hartwig, Sebastian Kalchgruber, Michael Fischer, Christoph Rauch, Markus Kaiser. Insbesondere möchte ich die beiden zuletzt Erwähnten hervorheben, die, ebenso wie meine Betreuer, mit umfangreichen Korrekturen und hilfreicher Kritik zum finalen Stand dieses Dokuments maßgeblich beitrugen. Ein zusätzlicher Dank geht an Markus, mit dem ich ein Büro teilen durfte. Wir hatten eine großartige Zeit.

Abschließend sei erwähnt, dass diese Arbeit nicht ohne die Zusammenarbeit mit der Firma Ascending Technologies GmbH möglich gewesen wäre. Mein herzliches Dankeschön an das Unternehmen und seine Mitarbeiter, die die Zeit, die ich mit dieser Arbeit verbracht habe, sehr angenehm in Erinnerung bleiben lassen.

Contents

1 Introduction

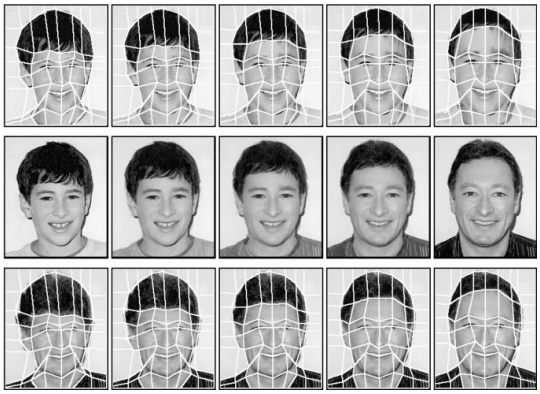
The motion picture *The Matrix* from 1999 features a scene in which Neo, the protagonist, spectacularly dodges bullets. Time gradually slows down and eventually freezes completely (Figure 1.1a). The camera starts a 360° rotation, pivoting the scene. Time accelerates and reverts back to its normal speed. This effect soon became known as the *Bullet Time* effect [Røs08, chap. 1]. Over the last few years visual effects like the Bullet Time effect have gained dramatically in both quality and number. Especially the use of image interpolation techniques has steadily increased. They are being applied in a wide variety of tasks, ranging from face morphing and slow motion effects to virtual video cameras (cf. Section 1.2). Just recently, the commercialised *freeD*™ technology from *Replay Technologies Inc.* led to impressive, never before seen shots of sport events like baseball, football, golf or even the olympic games (cf. [Inc13]). Until some years now, visual effects like these have been expensive and needed a lot of time and work in their making. For example, the camera rig that has been built for the very scene shown in Figure 1.1a consisted of over a hundred accurately planned, laser-positioned, calibrated, static cameras, placed in a green screen room [Røs08, chap. 1]. However, current research as seen in [ZKU+04, IS05, LLBM09] led to sophisticated solutions for achieving comparable effects with much less expensive setups. Nevertheless, many state of the art algorithms still require manual correction or additional user input.

This thesis aims to provide a low-budget, easy to use framework for generating state of the art visual effects. The goal of this work is to develop and implement a set of algorithms for image sequence interpolation in both space (one frame, multiple cameras) and time (multiple frames, single camera). The cameras used shall neither have to be calibrated nor accurately positioned, imposing as few constraints on the setup as possible.

1.1 Image Interpolation

Unfortunately, the term *image interpolation* is being used ambiguously in scientific papers. It often refers to the restoration of image information, examples ranging from resampling and hole-filling to the computation of superresolution images. In contrast, this paper uses *image interpolation* to refer to the computation of in-between images in an image sequence. Other common terms for this problem are *image sequence interpolation* or *multi-image interpolation* [Lip13]. When referring to the interpolation of

(a) Scene from the motion picture The Matrix showing the Bullet Time effect. Dozens of cameras film synchronously. To make full use of the expensive setup, the same shot was used several times in that movie.



(b) Face morphing. User specified point matches are linearly interpolated to transform one face into another. It consists of image warping (top, bottom) and blending (middle). [Wol98]

Figure 1.1: Image interpolation examples, showing its use in movies and face morphing.

different views (i.e. spatial interpolation), we will use the term *view interpolation* or *view morphing* as introduced by Seitz and Dyer [SD96]. Also, *free viewpoint video* is often used to underline a focus on spatial multi-camera interpolation [IS05].

Image interpolation usually consists of three parts: *Matching*, *warping* and *blending*. We refer to the latter two as *image morphing*. In image morphing, one first deforms a source image in such a way, that it gradually adjusts and finally matches a destination image. This can be done in both a *dense* (per-pixel) and *sparse* (selected features) manner. We will explain both approaches in chapters two and three respectively. By gradually warping each image towards its target, one obtains in-between images that mutually align (cf. Figure 1.1b). These synthesized images are then overlayed and blended, weighted by the distance to their sources. This results in convincing transitions between image pairs. Figure 1.1b shows an example of image morphing using mesh warping (cf. Chapter 3). The basic processing pipeline is outlined in Figure 1.2.

1.2 Related Work

Until the 1990s, computers had significantly advanced and eventually got powerful enough for sophisticated image processing. Image interpolation was on the rise. It had its first big breakthrough in 1992, when Beier and Neely [BN92] did a remarkable job for Michael Jackson's music video *Black or White*. People of different ages, genders and skin colors transform seamlessly into each other [Jac91]. They used manually selected line features to match faces between images and developed an algorithm to warp one feature

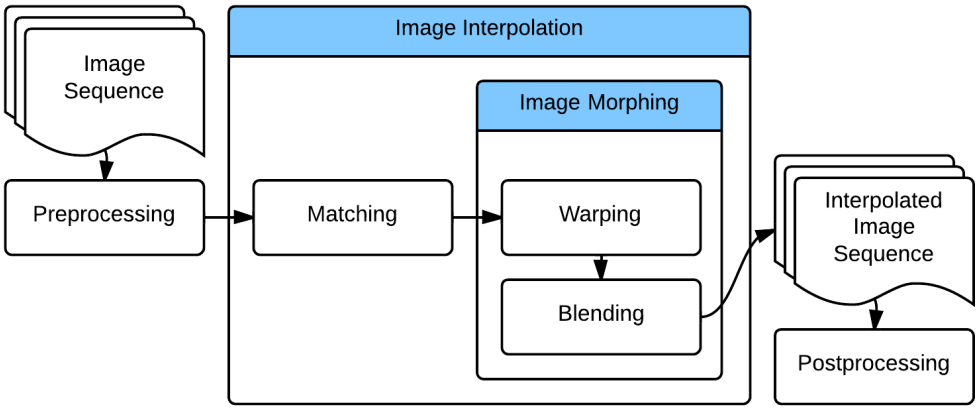


Figure 1.2: Basic image interpolation processing pipeline.

set to another. In the following years, more research was done in the area of feature based image morphing. Wolberg [Wol98] and more currently Vlad [Vla10] summarize the results. In 1993, Chen and Williams [CW93] presented an image synthesis procedure to generate novel in-between views of image pairs. They work with computer generated images exclusively, to be able to use the precomputed depth maps in order to warp images. Probably the most relevant work in the field of image interpolation is that of Seitz and Dyer [SD96]. They concentrate on view morphing of static scenes and propose a novel warping scheme using rectification for pre- and postwarping. They prove that their approach leads to realistic in-between views, as opposed to naive linear interpolation, which can lead to unpleasant deformations [SD95]. Fragneto *et al.* [FFR+12] improve upon Seitz and Dyer by adjusting the pre- and postwarping pipeline. They also introduce a method for homography interpolation without user interaction. Manning and Dyer [MD99] build upon [SD96] and incorporate change over time, leading to *dynamic view morphing*. They use foreground-background segmentation and matting to interpolate seperate layers individually. While Seitz and Dyer use a simple scanline matching algorithm (cf. [SD95]), both Avidan and Shashua [AS97] and afterwards Xiao and Shah [XS04] adapt tensors and tri-views to get more robust matching results. The next big step was done by Zitnick *et al.* [ZKU+04]. They generated one of the first free viewpoint video scenes. Their setup consists of a handful of static, calibrated cameras, positioned with small baselines[1]. For interpolation, they use layers, matting and a novel color segmentation-based stereo algorithm to estimate depth maps. Inamoto and Saito [IS05] and Replay Technologies Inc. [Inc13] go one step further and make use of 3d-registration and projection on top of layer segmentation. This technique is state of the art in free viewpoint video from static cameras. For dynamic scenes, Lipski *et al.* [LLBM09, Lip13] propose a novel dense matching scheme based on combined edgelet matching and color

[1]The baseline refers to the distance between the optical centers of different cameras.

segmentation. Their algorithm can be used on uncalibrated and handheld videos in both space and time. For interpolating images with minor variations (i.e. consecutive video frames), Werlberger *et al.* [WPUB11] and Chen and Lorenz [CL12] suggest the use of optical flow methods. Linz *et al.* [LLM10] introduce a selection-based morphing scheme on per-pixel basis, effectively avoiding artifacts that can occur with standard warping and blending methods. Their work is considered state of the art for dense image interpolation. For photo and video exploration, Ballan *et al.* [BBPP10] take a different route and first render a static 3d background model. They segment the object of interest and overlay an in-between view using billboards and matting.

In this thesis, we join the work of Chen and Lorenz [CL12] for temporal and the work of Seitz and Dyer, Wolberg and Fragneto *et al.* [SD96, Wol98, FFR+12] for spatial image interpolation. More sophisticated methods as presented in [Lip13] will not be covered in this work.

1.3 Outline

This thesis has its focus on optical flow and mesh warping as tools for simple image interpolation. The algorithms are implemented in C++ using the open source vision library OpenCV and the graphics library OpenGL.

Having introduced the concept of image interpolation in general in this chapter, the second chapter of this thesis covers temporal interpolation. It describes the use of dense image matching techniques for temporal image sequence interpolation. Additionally, basic image warping techniques and the concepts of forward and reverse mapping are covered. Chapter three first compares different methods for feature based warping, before addressing sparse image matching. Various feature detectors, descriptors and matching procedures are presented, evaluated and further enhanced. Triangulation is covered as a bridge to mesh warping. The chapter closes presenting view morphing in connection with image rectification as introduced by Seitz and Dyer. The fourth chapter gives an overview of the developed framework and outlines some of the applied code optimization techniques. Finally, results and possible applications are shown. The thesis closes discussing the capabilities and limitations of the presented algorithms.

2 Temporal Interpolation

In this chapter, we introduce dense (i.e. pixelwise) correspondences between image pairs and their use in temporal image sequence interpolation. After comparing different algorithms for optical flow computation, block and per-pixel matching, we address general image mapping techniques and show how these can be applied in dense image warping.

Seitz and Dyer use a simple scanline algorithm to compute pixelwise correspondences between two rectified (cf. Section 3.4.1) images [SD96]. Zitnick *et al.* [ZKU+04] and Lipski *et al.* [LLBM09] apply more sophisticated methods including color segmentation and edgelet matching to achieve more precise correspondences. In this thesis, we concentrate on the use of optical flow as presented by Werlberger *et al.* [WPUB11] and Chen and Lorenz [CL12] as a simple and effective tool for generating high quality correspondences between images with small variations (e.g. consecutive video frames and small baseline images).

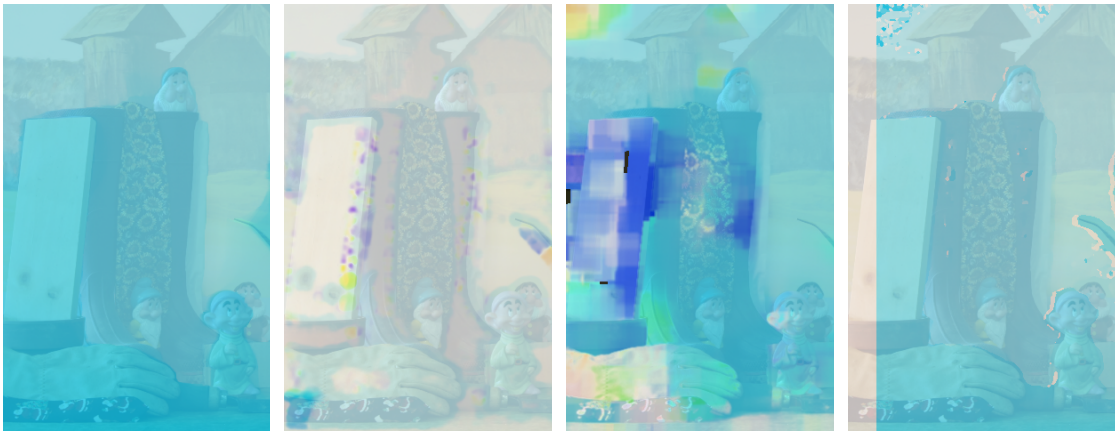
2.1 Optical Flow

Quote 2.1: Optical Flow

[The] optical flow is the distribution of apparent velocities of movement of brightness patterns in an image. Optical flow can arise from relative motion of objects and the viewer [...].

Horn, Schunk [HS81]

The flow field between two images can be interpreted as the relative displacement of pixels between two images. We compare three optical flow algorithms: Dual TV-L1 [SPMLF13], Farneback [Far03] and SimpleFlow [TBKP12]. Figure 2.1 visualizes exemplary results of flow fields, computed using OpenCV implementations with default parameters. We use the Middlebury flow color coding which maps angles and magnitudes of displacements to relative hue and saturation values [BSL+11]. As can be seen in Figure 2.1, Dual TV-L1 results in the most accurate correspondences. The Farneback flow has low computation costs, but suffers in quality. Whole areas are mismatched, leading to unpleasant local distortions and inaccurate movements in warped images. To exploit the full capability of SimpleFlow, its parameters need to be individually adjusted to each image sequence. This, however, does not meet our requirements for autonomous,



(a) Dual TV-L1 (b) Farneback (c) Simple Flow (d) SGBM

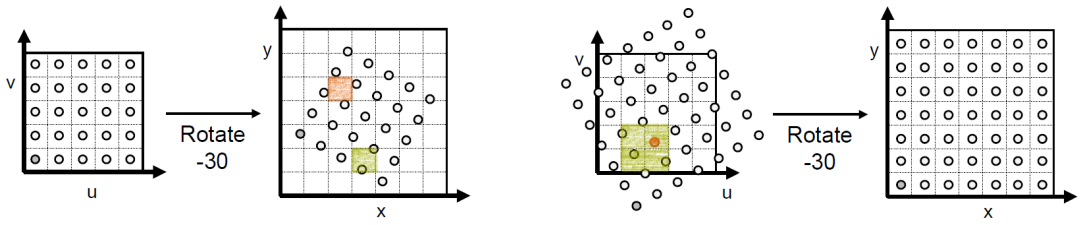
Figure 2.1: Dense correspondence evaluation on the Middlebury Dwarve dataset, comparing different algorithms using default parameters. Apart from Dual TV-L1, all algorithms need tedious parameter tuning for individual image sequences. For default settings, mismatches and holes occur in many scenes. We thus settle for Dual TV-L1 as primary flow estimation algorithm.

easy to use algorithms. With default parameters, the Dual TV-L1 algorithm yields best results for most scenes. It therefore forms the best foundation for further warping and blending steps. A more detailed evaluation is given in Section 5.2.

2.2 Block and Per-Pixel Matching

We also examined alternative approaches, the first being a scanline algorithm, comparable to that of Seitz and Dyer [SD95]. We adapted cross correlation to search for a closest match comparing small regions around each pixel. However, it soon became clear that this simple approach was not effective. While Seitz and Dyer rely on manually specified matchings of pixel intervals as guides, we naively search in the space of all image pixels. Due to the huge search space and ambiguous image regions, major artifacts and mismatches occur.

Another method we investigated is block matching. We use the OpenCV *Semi-Global Block Matching* [Hir08] (SGBM) implementation. As with SimpleFlow, SGBM works properly only with individually adjusted parameters for each image sequence. The implementation also requires rectified images. To be able to correctly rectify images, they need to fulfil a geometric relation (cf. Section 3.4.1), which is not always given, especially in the case of temporal image sequences. Block matching can also lead to unmatched regions, thus leaving holes and borders. Figure 2.1d shows an example exposing some of these problems. Summarizing, these are major drawbacks, as again, we look for algo-



(a) Forward mapping. The destination image is sampled. Holes (orange) and folds (green) can occur.

(b) Inverse mapping. The source image is sampled. Ambiguities (orange) can easily be resolved with neighbourhood interpolation (green).

Figure 2.2: The two types of mapping. [Fun00]

rithms that work on arbitrary images and with a minimum of user interaction. In the end, we settle for the optical flow approach.

2.3 Dense Image Warping

We first give the definition of an image.

Definition 2.1: Image

An *image* $\mathcal{I}$ is a mapping from a domain Ω to some color space Σ:

$$\mathcal{I} : (\Omega \subset \mathbb{R}^n) \to (\Sigma \subset \mathbb{R}^m) \tag{2.1}$$

We treat all images in this thesis as discrete, two dimensional RGB-images, thus $\mathcal{I} : \mathbb{N}^2 \to \mathbb{R}^3$. Image warping can be interpreted as a mapping for each pixel from a source image $\mathcal{I}_1$ to a target image $\mathcal{I}_2$ This mapping is in our case given by the optical flow $\mathcal{F}_{1,2} := \mathcal{F} : \mathbb{N}^2 \to \mathbb{R}^2$ from source to target image. We store flow maps as tensors, holding a two-dimensional displacement vector for every pixel position $\mathbf{p} = (x, y)^T$. There are two ways to apply such a mapping.

Definition 2.2: Forward Mapping

In a *forward mapping* approach, one iterates over every pixel $\mathbf{p} = (x, y)^T$ in the source image $\mathcal{I}$ and maps it to its corresponding position in the destination image $\mathcal{I}'$:

$$\mathcal{I}'(x + \mathcal{F}_x(x, y),\ y + \mathcal{F}_y(x, y)) = \mathcal{I}(x, y)\ . \tag{2.2}$$

As we are dealing with a discrete image domain Ω (i.e. pixels) on the one hand and displacements in $\mathbb{R}^2$ on the other, sampling artifacts can occur. Some destination pixels may be covered with multiple samples (folds, Figure 2.2a green square), others with none at all (holes, Figure 2.2a orange square). A solution to this problem is offered by the *backward mapping* approach (also often reffered to as *reverse* or *inverse mapping*).

Definition 2.3: Backward Mapping

In an *backward mapping* approach, one iterates over every pixel $\mathbf{p} = (x, y)^T$ in the destination image $\mathcal{I}'$ and assigns to it the value of its origin in the source image $\mathcal{I}$:

$$\mathcal{I}'(x, y) = \mathcal{I}(x + {\mathcal{F}_x}^{-1}(x, y),\ y + {\mathcal{F}_y}^{-1}(x, y)) \ . \tag{2.3}$$

Here, indistinct samples can easily be computed with bilinear or Gaussian interpolation of the surrounding source image pixels (Figure 2.2b). The OpenCV optical flow methods return a forward flow $\mathcal{F}_{1,2}$ from source to destination image. To apply backward mapping, one needs to find its inverse ${\mathcal{F}_{1,2}}^{-1}$. We do this by noticing that ${\mathcal{F}_{1,2}}^{-1} \approx \mathcal{F}_{2,1}$, as the flow is symmetric. Thus, using mutual flows we can apply Equation 2.3 to warp both images to its corresponding target. By linearly increasing each pixel's displacement from zero to its respective maximum value, one can achieve an effect of motion.

Definition 2.4: Linear Dense Warping

Let $\mathcal{F}$ be the flow from source image $\mathcal{I}_1$ to target image $\mathcal{I}_2$. A linear dense warping from source to target image is then given by

$$\mathcal{I}(x, y, t) = \mathcal{I}_1(x + t \cdot \mathcal{F}_x^{-1}(x, y),\ y + t \cdot \mathcal{F}_y^{-1}(x, y)) \ ,\ t \in [0, 1] \tag{2.4}$$

In particular $\mathcal{I}(x, y, 0) = \mathcal{I}_1(x, y)$ and for perfect flows $\mathcal{I}(x, y, 1) = \mathcal{I}_2(x, y)\ \forall x, y \in \Omega$.

Because the small movements in most optical flow based examples are hard to capture in still images, we refer to the accompanying framework for interactive results.

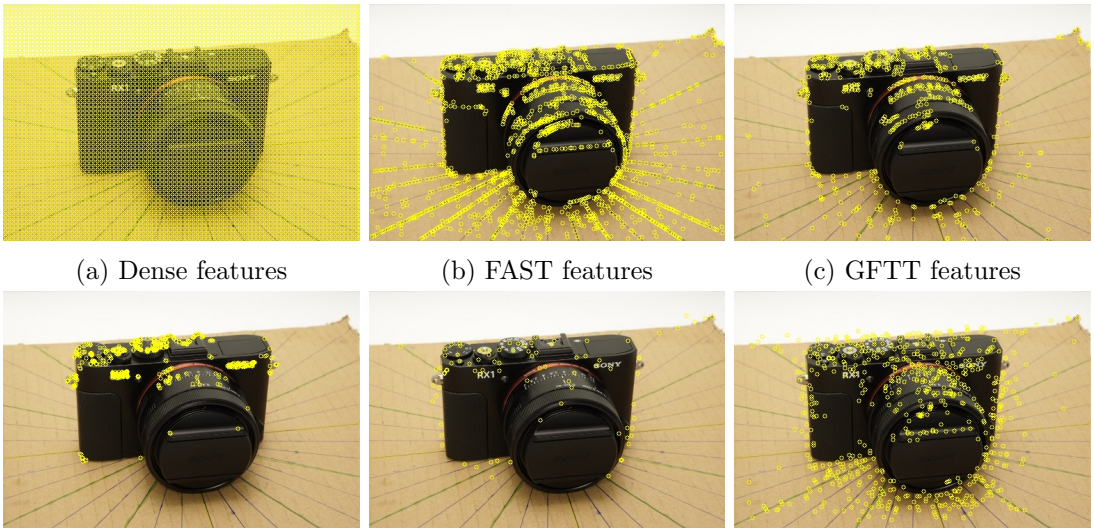
3 View Interpolation

The previous chapter showed how dense image correspondences can be used to generate warped in-between frames of image pairs using backward mapping. However, dense correspondence computation, especially the proposed optical flow methods, are limited to small variations between the input images. For high variances, arising from temporal differences or most notably changes in view, those methods fail to establish good or any correspondences at all. As the goal of this thesis is to provide an autonomous framework, robust enough not only to handle small variations but also large changes in view, the following chapter introduces sparse feature matching and view interpolation.

Wolberg and Vlad [Wol98, Vla10] evaluate common sparse matching and warping techniques, namely *Mesh Warping*, *Thin Plate Splines* and *Line Features*. As we want our methods to require a minimum of user interaction, Line Features are not practicable. Good line features are hard to detect and even harder to match automatically, as simple tests have shown. Thin Plate Splines interpolate globally, fitting a smooth surface to two dimensional data. This is not practicable either, as sharp edges get smoothed out. Imagine for example a square, where we define its four corners as points to interpolate. When we move one corner and deform the image accordingly, global interpolation will yield arcs and curves, rather than the original straight edges. Thus, we settle for Mesh Warping. Mesh Warping consists of three steps, that will be covered in the following sections: Finding and matching feature points (cf. Sections 3.1 to 3.5), creating a mesh by triangulating these points (cf. Section 3.6) and finally deforming the mesh to align to its target (cf. Section 3.7).

3.1 Feature Detectors

This section compares different feature detectors in order to extract robust and meaningful *features* or *keypoints* from an image, that can later be matched. A keypoint is the center of a small part of an image, that fulfills certain criteria, which are listed in Table 3.1 [TM07].



(a) Dense features (b) FAST features (c) GFTT features

(d) ORB features (e) SIFT features (f) SURF features

Figure 3.1: Comparison of some feature detectors provided in OpenCV.

Table 3.1: Keypoint Attributes

- **Local** - Occupies a small area, thus being robust to clutter and occlusions
- **Invariant** - Can be found across multiple images, regardless of geometric and phometric changes
- **Distinctive** - Can be uniquely described
- **Robust** - Noise, blur, quantization etc. do not destroy the feature description

A basic idea to find such features is realized in the *Harris Corner* detector [HS88]: It searches regions in an image that show high changes in more than one direction. This is a necessary and sufficient criterion for corners. The Harris Corner detector does this efficiently by computing gradients (more precisely the second moment Matrix $\mathbf{M}$) in each point and selecting those with high changes in two directions (large eigenvalues λ_1, λ_2 of $\mathbf{M}$). To understand why corners are particularly useful in feature matching, imagine a flat surface. This implies little to no intensity changes in any direction. A point on that surface can hardly be matched to a point in another image, as the information of its surroundings is completely inexpressive. This is contrary to the initial goal, to find good keypoints. There are many more strategies to detect "interesting" points and regions in an image. Figure 3.1 gives an overview of some feature detectors implemented in the OpenCV library. For our purposes, we want a set of features to fulfill the following, additional criteria:

Table 3.2: Matching Attributes

- **Distribution** - Not too close in order to get good meshes, but not too far apart which might lead to missed regions
- **Coverage** - Include most to all significant points, e.g. object outlines, corners, foreground and background

While the Dense and FAST detectors (Figures 3.1a and 3.1b) yield too many features, ORB and SIFT (Figures 3.1d and 3.1e) yield too few. All four candidates disagree with at least one of the criteria mentioned before, either for individual keypoints or for the set of all features. The SURF detector (Figure 3.1f) relies on integer approximated blob detection using the *Determinant of Hessian* [BETG08]. This leads to keypoints in areas of constant intensity, which are less robust than corners and irrelevant for good warping results. In the end, the *Good Features to Track* [TS94] (GFTT) proved to be a reliable and easily adjustable source of good features. It uses the Harris detector to find points of interest and filters a specified number of features according to their quality. Also, the distance between keypoints can be specified in terms of pixels, which enables the early prevention of too dense meshes.

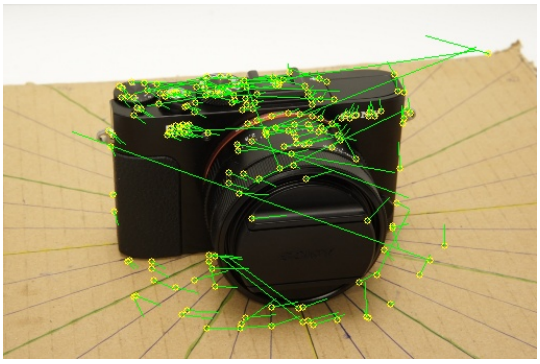
3.2 Feature Descriptors and Matching

After aquiring a set of keypoints, the next step is to match them across multiple images. This is done by first defining meaningful descriptions for these points. Such descriptors are basically arbitrary dimensional vectors holding information about a certain image region. This can include dominant orientations of gradients, intensity distributions, histograms or anything else, that might be unique for a certain feature point. Again, various approaches exist, but a detailed description is not in the scope of this thesis. We compare SIFT, SURF, BRIEF and BRISK descriptors, which are provided by the OpenCV library. Figure 3.2 shows some exemplary results. We evaluated the four methods comparing brute force matchings.

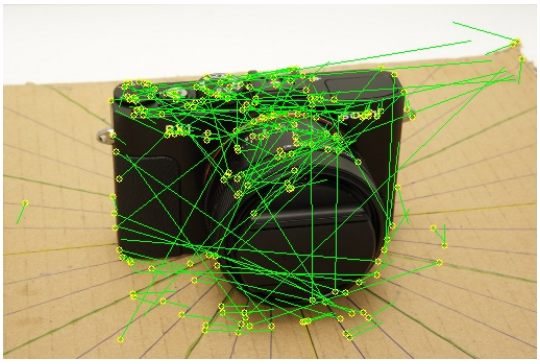
Definition 3.1: Brute Force Descriptor Matching

Let K_1 be a set of keypoints in a source image $\mathcal{I}_1$ and K_2 a set of keypoints in a target image $\mathcal{I}_2$. Let D_1, D_2 be their corresponding descriptors. Let further $M_{1,2}$ be an initially empty matching from K_1 to K_2, represented as a set of index pairs (i, j).

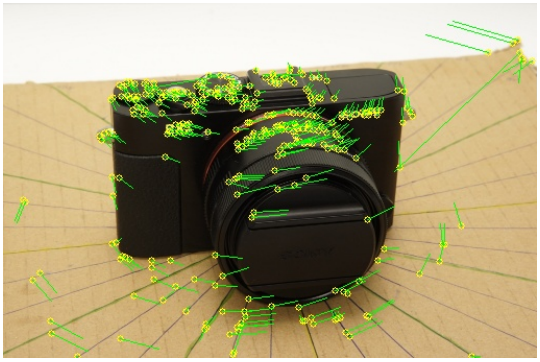
In a brute force approach, all possible combinations are listed and the best ones chosen, according to some metric. For descriptor matching, one computes the vector



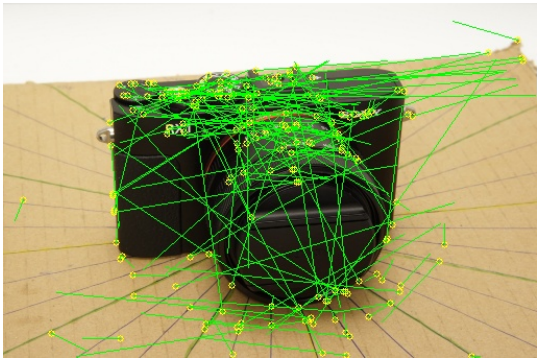
(a) BRIEF descriptor



(b) BRISK descriptor



(c) SIFT descriptor



(d) SURF descriptor

Figure 3.2: Comparison of brute force descriptor matchings. Green lines indicate corresponding matches. The images were taken at a rotational difference of 5° around the object of interest. Only source image is shown. Features were detected using Good Features To Track.

Definition 3.1: Brute Force Descriptor Matching

distances $\delta_{i,j} = \sqrt{\mathbf{d}_i \cdot \mathbf{d}_j}$ of all descriptor pairs $(\mathbf{d}_i, \mathbf{d}_j) \in D_1 \times D_2$. A match $\mathbf{m}_i = (i, j)$ for all keypoints $\mathbf{k}_i \in K_1$ is then added to $M_{1,2}$, where keypoint $\mathbf{k}_j$ has minimal descriptor distance $\delta_{i,j} = \delta_{min}$.

As was to be expected, the SIFT (Scale-Invariant Feature Transform) descriptor outmatches its competitors. It is explicitly designed to be robust against translation, rotation, scaling, noise and illumination changes [Low03, Low99]. It is also computationally more expensive. However, as image matching is done in an offline processing stage, this is none of our concern. The brute force matching on the other hand can be sped up without loosing much quality by using the FLANN (Fast Linear Approximated Nearest Neighbours) search algorithm. It efficiently finds an approximate best match for each descriptor [ML09].

3.3 Robust Matcher

Our first attempt towards an automatic sparse image matching algorithm was the *Robust Matcher*. The concept is based on the method from the *OpenCV Cookbook* [Lag11] and outlined in Algorithm 3.1.

Algorithm 3.1: Robust Matcher

1: **function** MATCH($\mathcal{I}_1, \mathcal{I}_2$)
2: $K_1, K_2 \leftarrow$ Detect SIFT keypoints
3: $D_1, D_2 \leftarrow$ Compute SIFT descriptors
4: $M_{1,2}, M_{2,1} \leftarrow$ Compute matches using FLANN
5: $M \leftarrow$ Discard non-symmetric matches
6: $M \leftarrow$ Filter epipolar inlier using RANSAC
7: **return** M
8: **end function**

After detecting and computing SIFT features and descriptors, FLANN is used to match each descriptor with its closest counterpart, resulting in matches $M_{1,2}$ and $M_{2,1}$ for features in images $\mathcal{I}_1$ and $\mathcal{I}_2$ respectively. Then, non-symmetric matches are discarded.

Definition 3.2: Symmetric Match

Let $M_{1,2}, M_{2,1}$ be mutual descriptor matchings. A match $\mathbf{m}_{i,j}$ of two descriptors $\mathbf{d}_i, \mathbf{d}_j$ is considered symmetric if both $\mathbf{m}_{i,j} \in M_{1,2}$ and $\mathbf{m}_{j,i} \in M_{2,1}$.

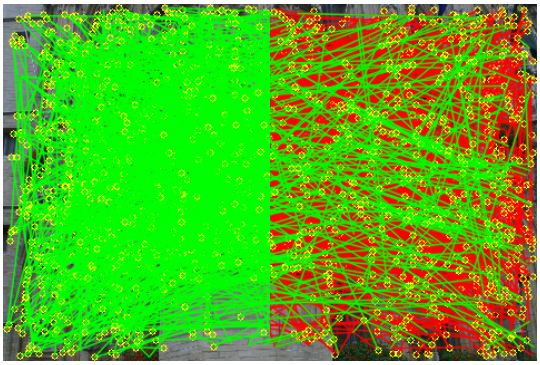
Afterwards, a set of "good" matches is selected by discarding those, that differ more than a certain threshold `maxDist` from the minimal descriptor-pair distance δ_{min}. We choose $\texttt{maxDist} = 0.5 \cdot (\delta_{max} - \delta_{min})$, to account for the better half. In a final filtering step, the *Random Sample Consensus* [FB81] (RANSAC) algorithm is applied to fit a geometric model to an optimal subset of inliers. This robustly eliminates false positive matches. The applied model is covered in Section 3.4. An example for the complete algorithm can be seen in Figure 3.3.

The Robust Matcher as presented in [Lag11] yields good results for well-textured images and small to medium baselines (camera rotations of up to 10° around a point of interest). However, two problems occur concerning both feature detection and matching, which will be addressed in the next section.

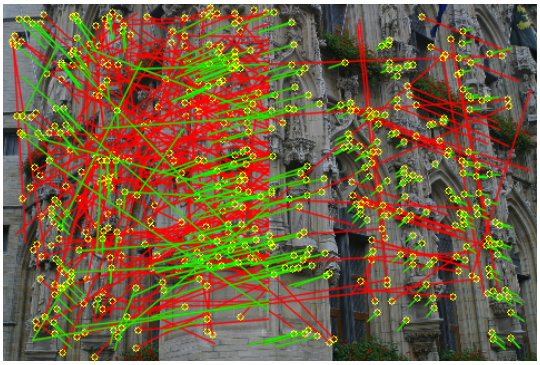
(a) Source image

(b) Target image

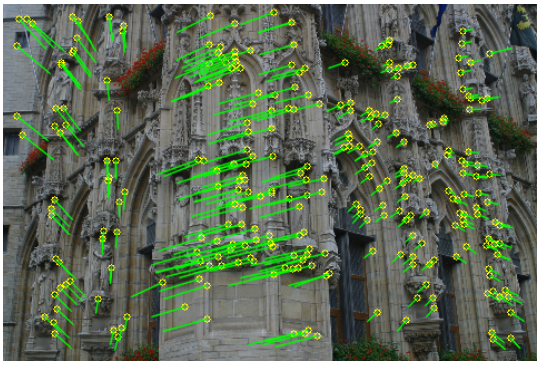
(c) Features detected with SIFT



(d) *Left*: Initial FLANN-based matching. *Right*: Symmetric matches



(e) *Left*: Matches after thresholding. *Right*: Robust outlier elimination using RANSAC



(f) The final matching visualized on the source image.

Figure 3.3: Step by step Robust Matcher pipeline. Features are shown in *yellow*, matches in *green*, outliers in *red*.

3.4 Epipolar Guided Matcher

The last section presented a robust algorithm to detect, match and filter sparse image features. The employed SIFT detector, however, while being robust in most scenarios, is not suitable for mesh warping as shown in Section 3.1. This can easily be solved by using the proposed GFTT features. As illustrated before, they make for a better foundation in both distribution and significance. The Robust Matchers' more serious weak point is the matching procedure itself. It establishes matches in the space of *all* possible feature combinations. Only afterwards, outliers are eliminated. We improve upon the Robust Matcher by restricting the search space as shown in the next section. As it appeared, Zhang *et al.* [ZDFL95] presented a very similar approach.

3.4.1 The Epipolar Constraint

Quote 3.1: Epipolar Geometry

The epipolar geometry is the intrinsic projective geometry between two views. It is independent of scene structure, and only depends on the cameras' internal parameters and relative pose. The fundamental matrix $\mathbf{F}$ *encapsulates this intrinsic geometry. It is a* 3×3 *matrix of rank* 2*. If a point in* 3*-space* $\mathbf{p}$ *is imaged as* $\mathbf{x}_1$ *in the first view, and* $\mathbf{x}_2$ *in the second, then the image points satisfy the relation* $\mathbf{x}_2\mathbf{F}\mathbf{x}_1 = \mathbf{0}$*.*

[HZ03, chap. 9, adjusted variables]

One can derive several other equations from this relation, the most important for our purposes being its left and right null spaces as given in Equation 3.2.

Definition 3.3: Epipolar Lines

Given the fundamental matrix $\mathbf{F}$ relating two images $\mathcal{I}_1$ and $\mathcal{I}_2$, let $\mathbf{x}_1 \in \Omega_{\mathcal{I}_1}$, $\mathbf{x}_2 \in \Omega_{\mathcal{I}_2}$ be two corresponding points such that

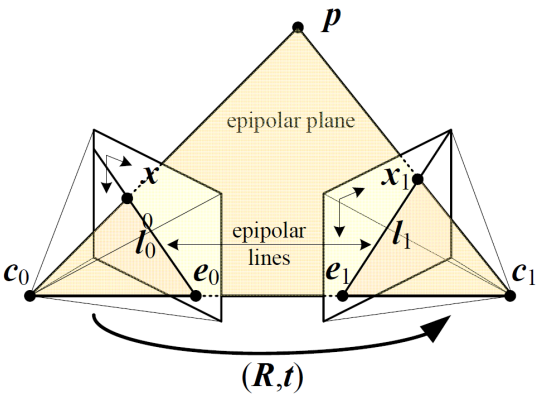
$$\mathbf{x}_2\mathbf{F}\mathbf{x}_1 = \mathbf{0}. \tag{3.1}$$

Then solving for the left and right nullspaces results in

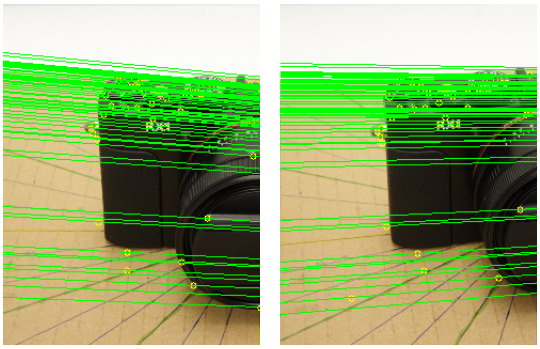
$$\mathbf{l}_2 = \mathbf{F}\mathbf{x}_1 \tag{3.2}$$

$$\mathbf{l}_1 = \mathbf{F}^T\mathbf{x}_2 \tag{3.3}$$

where $\mathbf{l}_2$ and $\mathbf{l}_1$ are called the *epipolar lines* for $\mathbf{x}_1$ and $\mathbf{x}_2$ respectively.



(a) Epipolar geometry for two camera centers $\mathbf{c}_0, \mathbf{c}_1$ and point $\mathbf{p}$. If $\mathbf{F}$ is known, the line $\mathbf{l}_1$ can easily be computed, which constrains the position of $\mathbf{x}_1$.



(b) Corresponding epipolar lines of matched features on the Sony dataset.

Figure 3.4: Epipolar geometry. *(a)*: Schematic, showing the epipolar geometry of a point p in two different views. [Sze11], *(b)*: Corresponding epipolar lines.

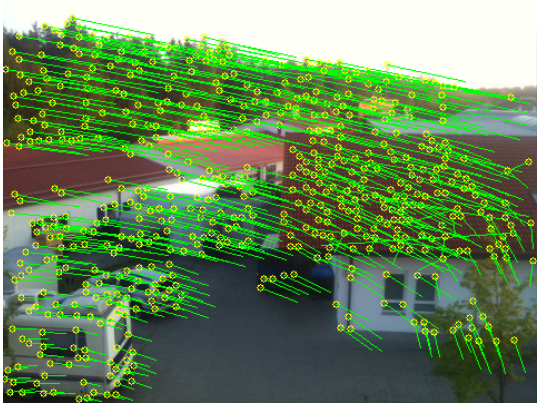
The projection $\mathbf{x}_1$ of a point $\mathbf{p}$ as seen in one view is thus by Definition 3.3 constrained to its epipolar line $\mathbf{l}_2$ in another view of the same scene (cf. Figure 3.4a). This effectively reduces the search space for matches from two image dimensions to one dimension along an epipolar line. From that, one can now devise a robust matching algorithm.

Algorithm 3.2: Epipolar Matcher

1: **function** MATCH($\mathcal{I}_1, \mathcal{I}_2$)
2: $M' \leftarrow$ Robust matching
3: $\mathbf{F} \leftarrow$ Estimate initial epipolar geometry
4: $M \leftarrow$ Guided matching using the epipolar constraint $\mathbf{F}$
5: $M \leftarrow$ Filter outliers
6: **return** M
7: **end function**

To estimate the fundamental matrix $\mathbf{F}$ one needs a set of sparse point matches. It should contain reliable and robust matches only. Last chapter's Robust Matcher (cf. Section 3.3) is perfectly fit for this task. After aquiring an initial set of matches, $\mathbf{F}$ is robustly estimated using the Random Sample Consensus (RANSAC) algorithm. RANSAC is an iterative statistic algorithm. In each iteration, a small sample of points is randomly selected from all potential matches and an estimate for $\mathbf{F}$ is computed. The points that satisfy Equation 3.1 within a certain threshold are chosen as inliers and a corrected model is estimated. These steps are repeated until convergence or reaching a fixed number of iterations. The method returns a set of inliers and a final estimation of the fundamental matrix $\mathbf{F}$. In a second matching step, we first detect GFTT keypoints and compute

(a) Robust matching. The matches are robust, but not suited for image interpolation.



(b) Epipolar guided matching. Good distribution, coverage and reliability of features form a solid foundation for further work.

Figure 3.5: Performance comparison of Robust and Epipolar Guided Matcher on the KimWest dataset.

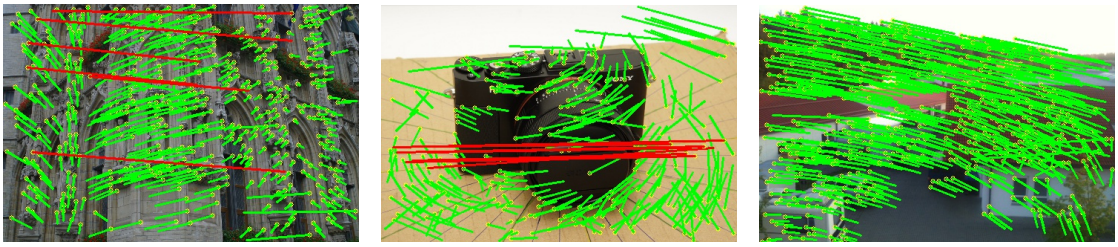
their SIFT descriptors. Best correspondences in terms of minimal descriptor distance along epipolar lines are searched using Equation 3.2. To adress noise and estimation errors, we extend each lines' width and accept matches within a certain distance threshold. All keypoints that lie outside this frame are not considered as potential matches. Additionally, we add a lower bound for the similarity of two descriptors, effectively eliminating uncertain matches early on. It is defined as the vector dot product of normalized descriptors and we take it to be 0.8, following Lowe [Low03].

We evaluate the presented method by comparing the Robust and Epipolar Matcher on different datasets. The quality criteria were again uniform distribution, sufficient coverage, meaningfulness and high reliability. For well-textured scenes, both algorithms show comparable results. The strength of our new approach shows in scenes with higher transition tilts[1], noise, blur, or inferior texture details in general. Figure 3.5 shows an exemplary result, comparing both matching algorithms on the KimWest dataset. However, while this approach leads to robust and especially well-distributed matchings, outliers can still occur. The next two subsections address this problem.

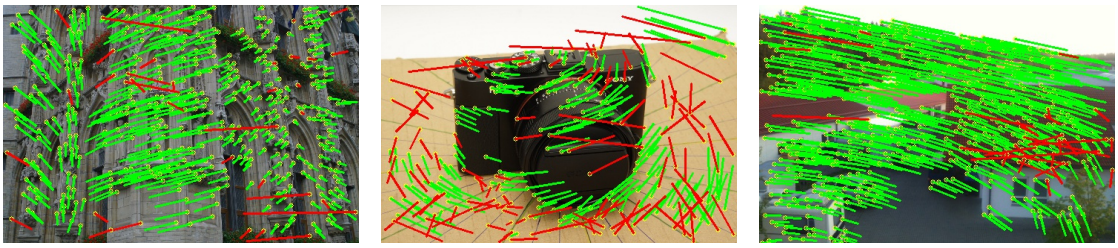
3.4.2 Playground Filter

We developed a global filtering procedure, that effectively eliminates obvious false positive matches. This is done using a k-Means clustering algorithm, that groups matches into keypoint-pairs of small, large and extreme displacements. The displacement is simply computed as the absolute distance between two keypoint positions. The idea

[1] A metric measuring the tilt between two views of the same scene (cf. [MY09])



(a) Playground Filter. Matches are globally filtered using clustering.



(b) Council Filter. Matches are locally filtered based on neighbourhood comparison.

Figure 3.6: Outlier elimination using the presented algorithms. *Green lines*: Inlier matches. *Red lines*: Classified outliers.

behind our clustering approach is to categorize matches based on their relative motion: Foreground motion is attributed to small displacements, background motion to large displacements. Extreme motions however are categorized as false matches and thus discarded. To prevent the loss of good matches by discarding a cluster that does not contain actual outliers, we introduce a lower bound for the separation between large and extreme clusters. It is defined as the relative distance between the centroids of two clusters. The following table lists all parameters and the chosen values in our implementation:

Table 3.3: Playground Filter Parameters

Variable	Value	Explenation
`maxIter`	1000	maximal number of iterations
`minEps`	0.001	minimal required centroid movement
`numClasses`	3	number of cluster classes
`minNormRatio`	0.6	lower bound for outlier cluster separation

Figure 3.6a shows some examples. While the presented filtering approach succesfully identifies obvious outliers (i.e. significantly different displacements), local false positives are not detected. The following section therefore presents a local filtering method based on neighbourhood comparison.

3.4.3 Council Filter

We propose a robust method to identify false positive matches with non-extreme displacements. Instead of globally grouping matches into clusters as done in the previous section, we compare subsets of matches in small regions around a candidate. Similar to a council, the neighbours vote for or against the candidate, according to certain criteria. If a match attains enough votes, it is accepted, otherwise it is rejected as an outlier. Table 3.4 lists the set of free parameters.

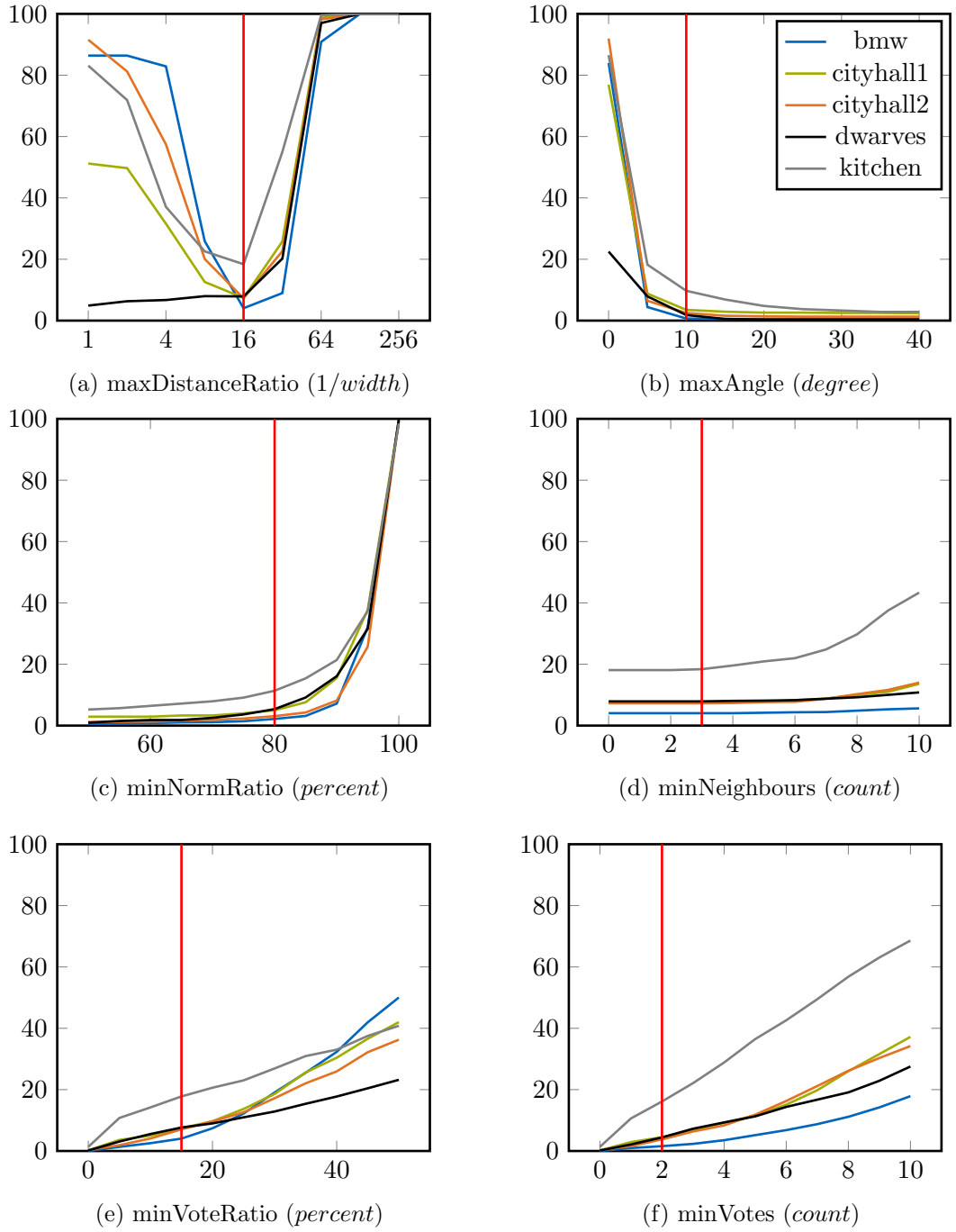
Table 3.4: Council Filter Parameters

Variable	Value	Explenation
`maxDistanceRatio`	1/16	neighbourhood radius, relative to image width
`maxAngle`	10	maximal angular difference in degree
`minNormRatio`	0.8	minimal relative displacement conformance
`minNeighbours`	3	minimal council size
`minVotes`	2	minimal number of attained votes
`minVoteRatio`	0.15	minimal ratio of attained votes
`nonQuorumPass`	`false`	accept as inlier if council is too small

We evaluate the algorithm's parameters by exporting raw and image data concerning the amount of outliers in relation to the total count of matches. This is done by fixing all but one parameter to roughly estimated default values, for all parameters. The choice of parameters is often one of balance. In our case, we choose that very value, for which most outliers are identified while simultaneously the amount of misclassifications is minimal. Conveniently, this value almost always is an extremum, either of the function itself or of its first derivative. If the results are ambiguous or unclear, we choose a parameter according to the worst performing example, in order to prevent false positive matches. This sometimes comes at the price of loosing true positive matches in other examples, which we accept to prevent unpleasant distortions in the final interpolation. Figure 3.7 shows plots of the evaluated parameters, which we will now explain in more detail.

A match is flagged as inlier if both keypoints are accepted by their respective councils. As first parameter, one can specify the radius of the covered neighbourhood around a candidate. We found, that an absolute value for the radius was not practicable as image down sampling is often applied in image matching to reduce image fidelity or computation time. As a consequence, `maxDistanceRatio` indicates the neighbourhood radius relative to the image width.

A councillor casts his vote for a candidate match if it satisfies two criteria. The displacements must coincide in both orientation and length. This is reflected in `maxAngle` and `minNormRatio`. The latter has already been introduced in previous algorithms and works in a similar way. Our evaluations show that it can be set more loosely than Lowe



(a) maxDistanceRatio ($1/width$)

(b) maxAngle ($degree$)

(c) minNormRatio ($percent$)

(d) minNeighbours ($count$)

(e) minVoteRatio ($percent$)

(f) minVotes ($count$)

Figure 3.7: Council filter parameter evalutaion. Red lines mark the chosen values. *Y-axis*: outlier ratio ($percent$). *X-axis*: free parameter values.

[Low03] did, classifying more true positives while still identifying most outliers. Nevertheless, we settle for the more conservative 0.8. The former, on the other hand, is to be chosen quite strictly. This, however, becomes clear after noticing that local angular differences above 10° only occur in cases of extreme camera motion.

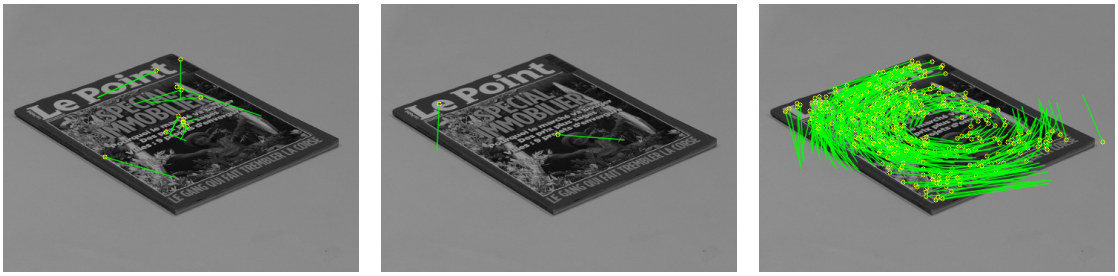
Furthermore, a candidate needs a minimum number of votes from its neighbours, both absolute and relative to the neighbourhood size, to be classified as an inlier. The absolute representation is particularly useful in situations with few neighbours. Consider a council of five members. According to `minVoteRatio`, a single vote would be sufficient to get accepted (i.e. $\lceil \texttt{minVoteRatio} \cdot \texttt{councilMembers} \rceil = \lceil 0.15 \cdot 5 \rceil = 1$). By introducing `minVotes`, one can require a minimum amount of confidence in terms of the absolute number of similar matches in a certain region. Nevertheless, we found that small neighbourhoods of four or less matches are not uncommon. For that we require at least two coinciding neighbours. However, if the council is too small to be able to form a proper sentence, then one can decide, whether a candidate should fail or pass by default. Figure 3.6b shows exemplary results, comparing the Robust and Epipolar Matcher's quality.

3.5 Affine-SIFT

The matcher presented in Section 3.4 yields very good matchings in most situations, but still fails in two extreme cases: Very wide baselines and low-textured images. To address this problem, we investigated ASIFT (*Affine-SIFT*) presented by Yu and Morel [MY09]. SIFT descriptors, which are robust against small viewpoint changes, fail to cover all six parameters of affine transformations. Namely, latitudal and longitudinal rotations are not considered. Yu and Morel compensate this lack of transformations *by simulating all possible affine distortions caused by the change of camera optical axis orientation from a frontal position* [MY09]. The ASIFT matching algorithm is outlined in Algorithm 3.3.

Algorithm 3.3: ASIFT Matcher

function MATCH($\mathcal{I}_1, \mathcal{I}_2$)
    $\mathcal{I}_{1,s}, \mathcal{I}_{2,s} \leftarrow$ Simulate all possible affine distortions
    $K_{1,s}, K_{2,s}, D_{1,s}, D_{2,s} \leftarrow$ Compute SIFT keypoints and descriptors for each simulation
    $M \leftarrow$ Match all keypoints using FLANN
    $M \leftarrow$ Threshold matches
    $M \leftarrow$ Remove identical matches
    $M \leftarrow$ Remove ambiguous matches
    $M \leftarrow$ Filter inlier using ORSA
    return
end function



(a) Robust Matcher (b) Epipolar Matcher (c) ASIFT Matcher

Figure 3.8: ASIFT evaluation by comparing different algorithms. The input image pair has a very high transition tilt. Only ASIFT is capable of handling such distortions.

Yu and Morel first simulate the sampled affine distortions and compute SIFT keypoints and descriptors for each such layer of simulation. Then, these descriptors are matched for all combinations of layers, leading to a vast amount of matches. As SIFT generally leaves behind false matches, "ASIFT, by comparing many pairs, can therefore accumulate many wrong matches" [MY09]. To filter outliers, Yu and Morel first eliminate identical matches, that is matches with both keypoints at a maximal distance of one pixel diagonal. Second, they eliminate ambiguous keypoints, that is keypoints which are matched multiple times in another image. Finally, they apply ORSA (*Optimized Random Sampling Algorithm*) to eliminate outliers based on epipolar geometry. ORSA is comparable to the previously presented RANSAC method, but considered to be more reliable and more robust to outliers. Unfortunately, no implementation of ORSA was available. We thus remained with the RANSAC approach. Because RANSAC yields less reliable results, we extended the filtering pipeline by our Playground and Council filters. We additionally pre-filter good matches according to a mean threshold, as presented in in Section 3.3.

The results show what we anticipated (cf. Figure 3.8): ASIFT works great on wide baseline image pairs and on images, where our Epipolar Matcher cannot find enough features. Nevertheless, ASIFT matches with small displacements are often inaccurate due to the huge search space of all sampled affine parameters. Thus, we propose to use ASIFT only as an alternative in those situations in which the Epipolar Matcher fails.

3.6 Triangulation

The last sections showed how to reliably find and match feature points between two images. We now explain how to use these features as control points and gradually warp one image towards another by interpolating between feature positions. To render the warped image, we form a triangle mesh passing through all interpolated points. The

image is then mapped onto the mesh surface as texture. The following sections first detail the utilized triangulation approach, then cover the extrapolation of matches in unmatched regions and finally specify the warping procedure itself.

3.6.1 Delaunay Triangulation

Bowyer [Bow81] presents a simple method to compute *Delaunay Triangulations*. Delaunay Triangulations, in contrast to other concepts, maximize all inner angles of the triangles in a triangulation. This leads to more regular triangles and tends to avoid acute ones. The Delaunay Triangulation is an iterative algorithm, that adds points one after another to a valid triangulation. The complete procedure is shown in Algorithm 3.4.

Algorithm 3.4: Bowyer Delaunay Triangulation

```
1:  function TRIANGULATE(points P, rectangle R)
2:      triangles T ← ∅
3:      T ← t_1, t_2 dividing R
4:      for p in P do
5:          INSERT(p, T)
6:      end for
7:      remove initial triangles t_1, t_2 from T
8:      return T
9:  end function

10: function INSERT(point p, triangles T)
11:     edge multi-set E ← ∅
12:     for t in T do
13:         if p lies within the circumcircle of t then
14:             remove t from T
15:             store edges of t in E
16:         end if
17:     end for
18:     for e = {a, b} in E do
19:         if e is unique in E then
20:             add new triangle (a, b, p) to T
21:         end if
22:     end for
23: end function
```

One starts with an initial, valid triangulation. It is implicitly given by the user as rectangular bounding box, that is split along the diagonal into two triangles. To insert a new point, the naive approach searches for all triangles, whose circumcircles enclose that point. These triangles are removed and their edges stored. New triangles are then

constructed for all unique edges (i.e. those stored only once) that previously have been removed. The circumcircle of a triangle is given by Definition 3.4.

Definition 3.4: Circumcircle

The circumcircle of a triangle $\mathbf{t} = (\mathbf{a}, \mathbf{b}, \mathbf{c})$ in $\mathbb{R}^2$ is the unique circle, that passes through all three triangle corners. Its center $\mathbf{c} = (c_x, c_y)$ can be computed according to the following equation:

$$c_x = \frac{1}{2k} \cdot \left(\frac{\mathbf{a}^T\mathbf{a} - \mathbf{c}^T\mathbf{c}}{b_y - c_y} - \frac{\mathbf{b}^T\mathbf{b} - \mathbf{c}^T\mathbf{c}}{a_y - c_y} \right)$$
$$c_y = \frac{1}{2k} \cdot \left(\frac{\mathbf{b}^T\mathbf{b} - \mathbf{c}^T\mathbf{c}}{a_x - c_x} - \frac{\mathbf{a}^T\mathbf{a} - \mathbf{c}^T\mathbf{c}}{b_x - c_x} \right)$$

where

$$k = (a_x - c_x) \cdot (b_y - c_y) - (a_y - c_y) \cdot (b_x - c_x)$$

Its radius r is given by the distance from $\mathbf{c}$ to any of the corners:

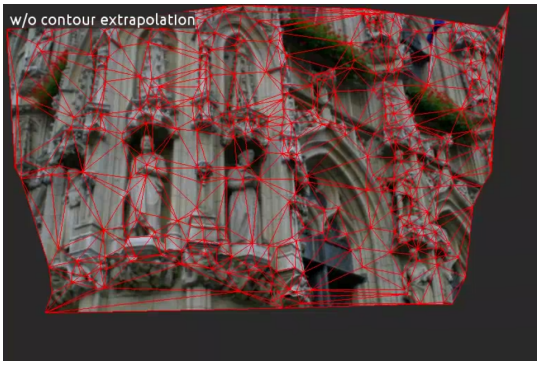
$$r = \sqrt{(c_x - a_x)^2 + (c_y - a_y)^2}$$

Whether a point $\mathbf{p} = (p_x, p_y)$ lies within the circumcircle of a triangle can then easily be computed by comparing the circumcircles radius r and the distance from $\mathbf{p}$ to its center $\mathbf{c}$. An exemplary triangulation is shown in Figure 3.9a. It can be observed, that unmatched regions don't have any texture, as there are no points to triangulate. This is especially present at the image borders. The next section outlines a solution to this problem via extrapolation of border matches.

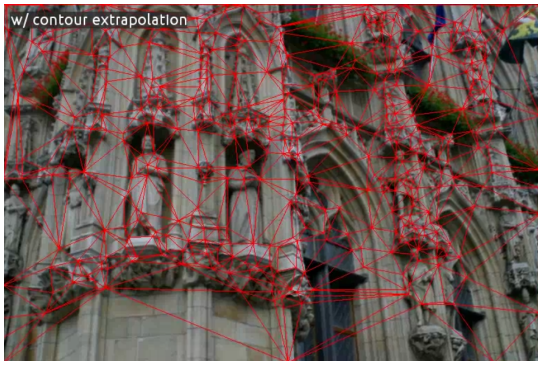
3.6.2 Border Extrapolation

Because of the nature of Bowyer's algorithm, only the input points themselves are being triangulated. This leaves unmatched regions where no texture can be mapped to. Simply adding the four image corners and estimating their displacements based on nearest neighbours results in overlapping triangles. This in turn introduces unpleasant artifacts during interpolation. We tried to solve this overlapping by re-triangulating the extended set of points in every step. However, this also proofed to be misleading. Different triangulations introduce unpleasant discrepancies and again artifacts between two interpolation steps.

Instead, after some consideration and evaluation, we developed a novel extrapolation approach that searches the *contour* of a triangulation and projects it onto the bounding rectangle's edges. We define it as follows.



(a) Delaunay triangulation without the initialization rectangle.



(b) Same example, but with border extrapolation using radial contour projection.

Figure 3.9: Triangulation of matches for the cityhall dataset. *Left*: Without border extrapolation. *Right*: With border extrapolation.

Definition 3.5: Point Set Contour

Let P be a set of points in $\mathbb{R}^2$. Let further T be a valid delaunay triangulation of P, that has been initialized with points P' and triangles T'. The contour of P then is the set of points $C \subseteq P$, that have at least one edge to any $p \in P'$.

In contrast to the convex hull, the contour is a more elaborate outer boundary of a point set with the intention to include even concave sections (cf. Figure 3.10). This leads to a denser and more accurate extrapolation. We find the contour by noticing, that all non-contour points form a wheel-graph with their respective neighbours. By traversing the neighbourhood-induced path, one can decide if a point lies on the contour by a simple test for cycles. After a set of contour points is aquired, we project these points onto the rectangle's edges. This can be done using either orthogonal (Figure 3.10a) or radial (Figure 3.10b) projection. For both methods the cardinal direction (above, below, left, right) relative to the rectangle center has to be known. It can be computed as shown in Definition 3.6.

Definition 3.6: Cardinal Directions

Given a point $\mathbf{p} = (x, y)^T$ in rectangle R with center $\mathbf{c}$ and dimensions $\mathbf{d} = (d_x, d_y)^T$. Let $r = d_y \cdot {d_x}^{-1}$. Then the directions of diagonals $\mathbf{a}_1 = (1, -r)$ and $\mathbf{a}_2 = (-1, -r)$ form a two dimensional basis. The cardinal direction can be found solving for a linear combination $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$, where $\mathbf{b} = \mathbf{p} - \mathbf{c}$. A closed solution reads as follows:

$$\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)^T = (\det \mathbf{A})^{-1} \cdot (\det \mathbf{B}_1, \det \mathbf{B}_2)^T$$

Definition 3.6: Cardinal Directions

where

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_1 & \mathbf{a}_2 \end{pmatrix}, \mathbf{B}_1 = \begin{pmatrix} \mathbf{b} & \mathbf{a}_1 \end{pmatrix}, \mathbf{B}_2 = \begin{pmatrix} \mathbf{b} & \mathbf{a}_2 \end{pmatrix}$$

Point $\mathbf{p}$ lies in the northern rectangle section, if both $\mathbf{x}_1, \mathbf{x}_2 \geq 0$, it lies west if $\mathbf{x}_1 < 0$ and $\mathbf{x}_2 \geq 0$, vice versa for south and east.

Orthogonal projection can be achieved by simply clipping the corresponding coordinate, e.g. a northern point gets its y-coordinate set to zero, a southern point to d_y. For radial projection, one has to calculate the intersection of line $\overline{\mathbf{cp}}$ and the corresponding edge.

Definition 3.7: Line-Line Intersection

The point of intersection $\mathbf{p}$ of two non-parallel lines $\mathbf{l}_1 = \overline{\mathbf{p}_1\mathbf{p}_2}$ and $\mathbf{l}_2 = \overline{\mathbf{p}_3\mathbf{p}_4}$ is given by

$$\mathbf{p} = \mathbf{p}_1 + t \cdot \mathbf{n}_1$$

where

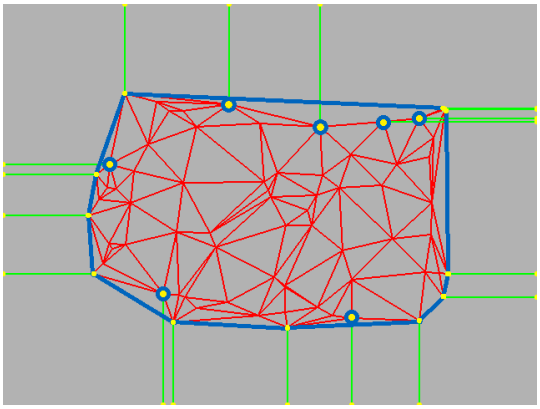
$$t = \det \mathbf{D} \cdot (\det \mathbf{N})^{-1}, \quad \mathbf{N} = \begin{pmatrix} \mathbf{n}_1 & \mathbf{n}_2 \end{pmatrix}, \quad \mathbf{D} = \begin{pmatrix} \mathbf{d} & \mathbf{n}_2 \end{pmatrix},$$
$$\mathbf{n}_1 = \mathbf{p}_2 - \mathbf{p}_1, \quad \mathbf{n}_2 = \mathbf{p}_4 - \mathbf{p}_3, \quad \mathbf{d} = \mathbf{p}_3 - \mathbf{p}_1$$

Two lines are parallel if and only if $\det \mathbf{N} = 0$.

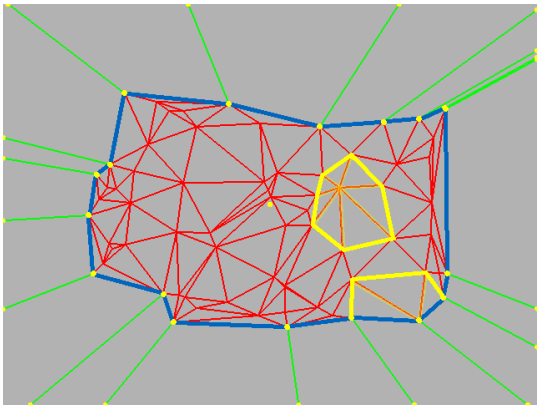
To extrapolate the missing displacement information for the projected points, we simply assign the motion between their contour counterpart and its match as a rough estimate. Finally, we re-triangulate the complete set of all points, including the extrapolated border points, with a slightly larger initial rectangle. Figure 3.9 shows a comparison with and without the extrapolation of matches on the image border.

3.7 Sparse Image Warping

The last sections showed, how to compute sets of matched points and according triangulations for a pair of images. To warp one image towards another, the simple mesh warping approach now linearly interpolates matches, while the image texture is overlayed on the triangulated mesh and anchored to the triangle corners. This is done in a forward mapping manner as shown in Equation 2.2, only that not every single pixel but a set of points are processed:



(a) Orthogonal projection of contour points onto the the image borders.
Blue lines: Convex hull of all points.
Blue circles: Outer points not being covered.



(b) Radial projection of contour points onto the image borders. *Blue lines*: Proposed contour of the shown triangulation.
Yellow lines: Two induced subgraphs for a contour and a non-contour point.

Figure 3.10: Extrapolation of matches on the image border using contour projection. The convex hull and our proposed contour are compared, shown in blue.

Definition 3.8: Linear Point Warping

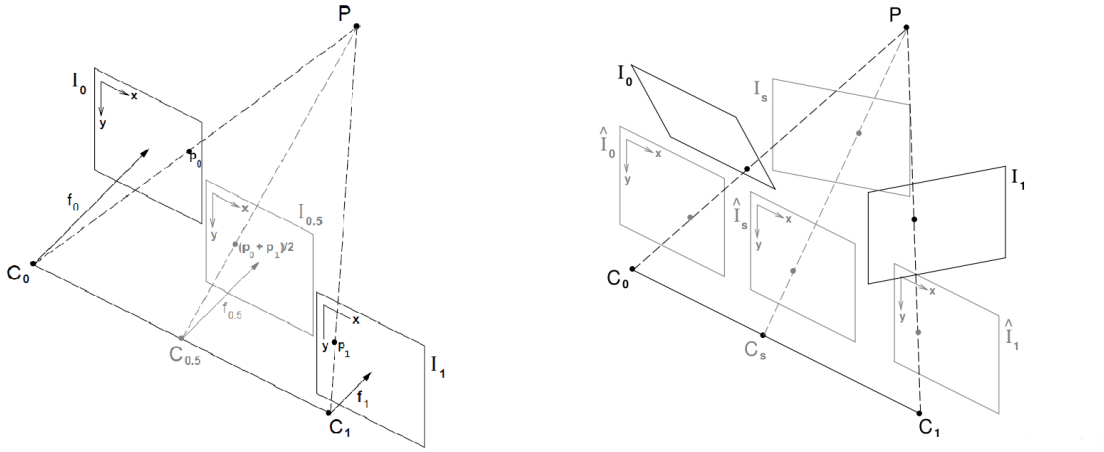
The linear interpolation of two points $\mathbf{p_1}$ and $\mathbf{p_2}$ is given by

$$\mathbf{p}(t) = \mathbf{p_1} + t \cdot (\mathbf{p_2} - \mathbf{p_1}) \ , \ t \in [0, 1] \tag{3.4}$$

However, Seitz and Dyer show that applying Definition 3.8 can result in unrealistic in-between views [SD95]. They propose a pre- and postwarping method, that was further enhanced by Fragneto *et al.* [FFR+12] and is covered in the next section.

3.7.1 View Morphing

Image morphing is able to produce plausible transformations between arbitrary images. Previous sections showed, how to apply linear interpolation of image correspondences to achieve such transformations. However, when dealing with images that show the same scene from different views, linear interpolation can lead to deformations and shape-distortions. These effects are especially present for rotations, as can be seen in Figure 3.12. Seitz and Dyer introduce rectification, homography interpolation and derectification as pre- and postwarping steps, that guarantee physically valid and thus shape-preserving interpolations. They prove their claim and name this adapted image interpolation *View Morphing* [SD96].



(a) Warping by point-wise linear interpolation is shape-preserving, iff the images lie in parallel planes. Otherwise, distortions and deformations are introduced.

(b) Images lying in arbitrary planes are first projected onto parallel planes, on which linear interpolation results in physically valid in-between views. After warping, they are backprojected into the non-rectified space.

Figure 3.11: Illustrations depicting the concept of view morphing [SD96]

Image rectification is closely coupled to the epipolar geometry introduced in Section 3.4.1. Its purpose is to transform two images in such a way, that their epipolar lines align horizontally. Thus all correspondences lie on horizontal lines, too, and their interpolation is indeed linear. Today, established algorithms for image rectification exist. Most of them additionally minimize image distortions by optimizing an according cost function. So does the OpenCV implementation based on Hartley [Har99] that we employ to use. The rectification procedure yields two homographies which, applied to two input images, result in horizontally aligned features. Seitz and Dyer now propose the following scheme: First project two images onto parallel planes (*prewarping*), interpolate matched feature positions by means of Definition 3.8 and finally backproject the warped images into the non-rectified space (*postwarping*). Figure 3.11 exemplary illustrates these steps. The question remains how to backproject the interpolated positions from rectification to world space. While Seitz and Dyer rely on user guided homography interpolation, Fragneto *et al.* present a simple interpolation method, that is given in Definition 3.9.

Definition 3.9: Homography Interpolation [FFR+12]

Given two rectification homographies $\mathbf{H_1}, \mathbf{H_2}$. Let $\mathbf{H} = \mathbf{H_1}^{-1}\mathbf{H_2}$. An intermediate homography at $t \in [0, 1]$ is then given by

$$\mathbf{H}(t) = \mathbf{H_1}\mathbf{H}^t = \mathbf{H_1} \cdot exp(t \cdot log(\mathbf{H})) \ ,$$

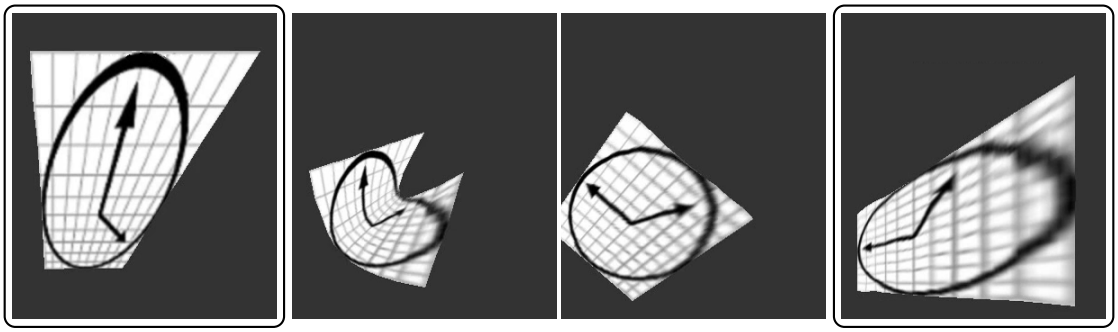


Figure 3.12: View morphing of the Clock dataset [SD96]. *From left to right*: Source image, unrectified interpolation, rectified interpolation, target image. The interpolations are computed at $t = 0.5$.

We can now summarize the complete warping process:

Algorithm 3.5: View Morphing

function WARP(images $\mathcal{I}_1$, matched points P_1, P_2, fundamental matrix $\mathbf{F}$, t)
  $\mathbf{H}_1, \mathbf{H}_2 \leftarrow$ STEREORECTIFYUNCALIBRATED$(P_1, P_2, \mathbf{F})$
  $\mathbf{H}_t \leftarrow$ interpolate homographies (Definition 3.9)
  prewarping: apply $\mathbf{H}_1, \mathbf{H}_2$ to P_1, P_2 respectively
  warping: $P_t \leftarrow$ linearly interpolate matches (Definition 3.8)
  postwarping: reproject P_t using $\mathbf{H}_t^{-1}$
  $\mathcal{I}_t \leftarrow$ draw textured triangle mesh at new vertex positions
  return $\mathcal{I}_t$
end function

Figure 3.12 demonstrates the effect of using pre- and postwarping. The comparison clearly shows, that the additional rectification step preserves straight lines and avoids unrealistic deformations. Nevertheless, view morphing has marginal to no visual effects on scenes under translation or small rotations. In these scenarios, the paths that points take during warping can be linearly approximated. The rectification process then won't introduce any perspective transformations. Consequently, pre- and postwarping do not have the huge impact we hoped them to have on real world examples, but only on a certain class of datasets as described above. Nevertheless, even unsuitable scenes seem more pleasant and smoother using view morphing. Extrapolated matches, however, sometimes fail the rectification process and have to be treated with caution.

4 Implementation Details

We implemented all presented algorithms in C++ using Eigen, OpenCV, OpenGL and the GLSL shader language. We ran all tests on an Intel Quad-Core i7 3.4 GHz machine with an Intel HD 4000 graphics chip and 16 GB DDR3 RAM. Establishing and storing dense matches between two images of width 480 px using Dual TV-L1 takes about one second. Sparse matches for the same input usually take about half a second. However, matching time was never an issue, as all data is precomputed and stored on the harddrive. After buffering all necessary files, the actual image interpolation can be explored in real time and Full HD. The following sections first give an overview of the framework in general, then outline some optimization approaches and finally cover the utilization of graphics hardware, which enables the interactive real time exploration in the first place.

4.1 Framework Overview

We built the framework in an object oriented and modular way. All implemented algorithms are encapsulated in classes that conform to certain interfaces and can therefore easily be exchanged. For example, all filter classes conform to the same `SparseMatchFilter` interface. In the same way, all feature matching implementations follow the `SparseMatcher` interface and likewise do all dense matcher classes. We also offer a set of `ImageInputStreams` to read different sources like image sequences or video files. We employ YAML-files (comparable to XML, but more human readable) as an easy to use interface. One can specify the input sources, different matcher algorithms to be used and combinations of inputs and algorithms to be computed and displayed. The main `BatchMorpher` class takes the path to such a YAML-file and buffers all necessary data. For dense matchings, this data consists of binary floating point maps of pairwise flow fields. Sparse matchings are stored as lists of point-wise matches, triangle indices to that matches, extrapolated matches with corresponding triangles and rectification homographies. If during buffering some matching files can not be found by the `BatchMorpher`, it applies the specified algorithms to automatically compute, store and reload missing matchings.

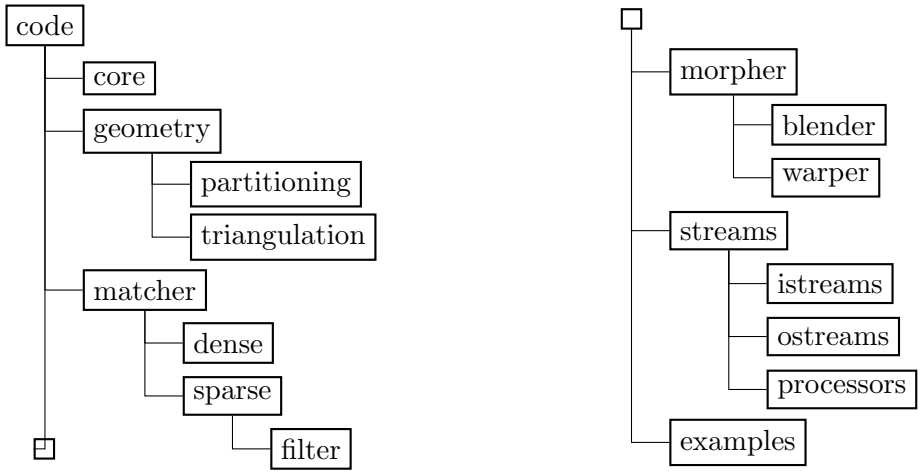


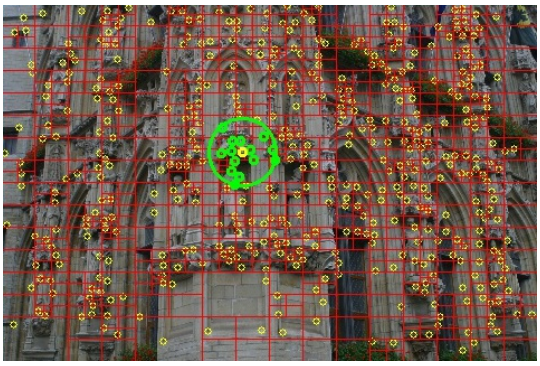
Figure 4.1: Framework structure. All code files are grouped into separate categories.

4.2 Code Optimization

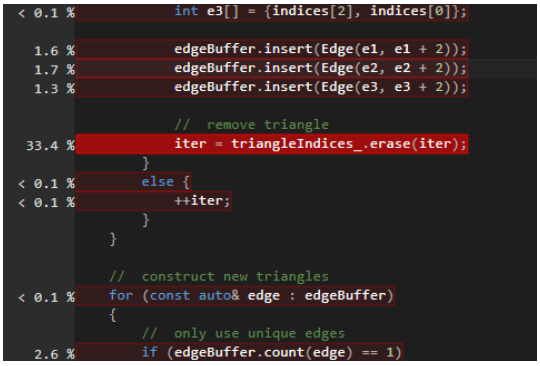
Despite the fact, that computation time for matchings was not critical, we adapted several optimization techniques to allow for faster evaluation, especially in trial and error scenarios. One such technique are quadtrees. We use this space-partitioning data structure to efficiently find elements in a given region (cf. Figure 4.2a, green circle). It is applied in the Council Filter presented in Section 3.4.3. We also use the *Visual Studio Analyzer* to find critical lines of code. An interesting case can be seen in Figure 4.2b, where the `insertPoint`-method in the Delaunay Triangulation implementation consumed over a third of overall computation time. After analyzing the code, it was obvious that the element-remove operation provided by the STL `vector`-class caused the overhead. Instead of erasing triangles one by one, we now store all triangles, that are not erased in a seperate list. The graphic shows, that this approach yields much faster execution times.

4.3 Hardware Acceleration

The interactive realtime image interpolation is made possible by using hardware acceleration. All sparse matching data is pre-loaded into RAM, while dense matchings and all images are buffered directly as textures in the graphics card's VRAM. Dense, pixel-based image warping is implemented solely in a GLSL fragment shader. Sparse image warping, based on features and triangle meshes, is impemented in two stages. We first compute the interpolation of feature positions with pre- and postwarping on the CPU using efficient OpenCV and Eigen matrix operations. Rendering is then accomplished utilizing OpenGL, specifying vertices and according texture coordinates for all



(a) Quadtree accelerated neighbourhood search on the Cityhall dataset. The image is partitioned into rectangles. An upper bound for the number of elements per region is specified. Regions with too many elements are further divided into subregions. The name *quadtree* results from the tree datastructure, in which each node holds four children.



(b) The Visual Studio Analyzer was used to find inefficient lines of code. For example, over one third of overall execution time has been spent in the triangulation method, more specifically in the STL `vector` function `erase`. After minor refactoring, this could be improved to not even be measurable any more ($< 0.1\,\%$).

Figure 4.2: Optimization techniques used in the implementation. These include algorithmic optimization (a) and code analysis (b).

interpolated points.

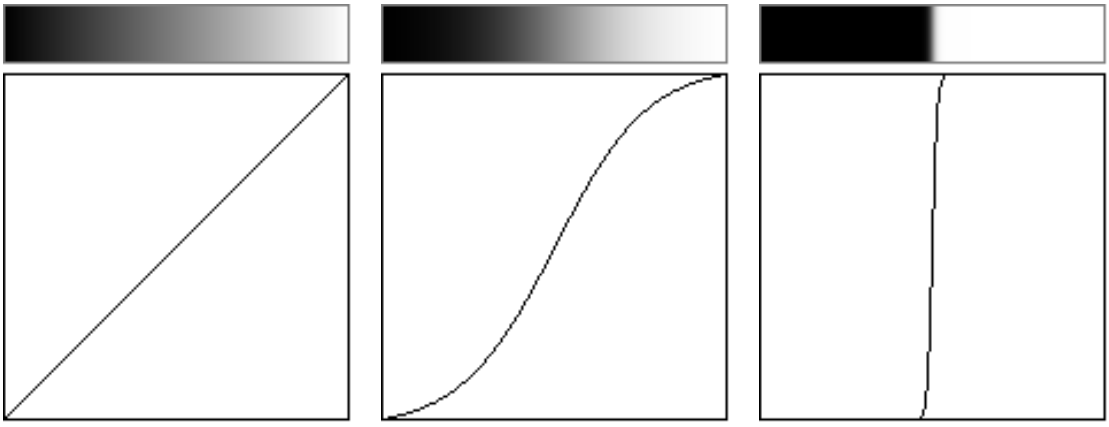
The final step of image interpolation has not been covered in more detail yet: Image Blending. Two images, warped to align each others feature points, are overlayed and cross-dissolved according to Definition 4.1.

Definition 4.1: Image Blending

Given two warped images $\mathcal{I}_1(t)$ and $\mathcal{I}_2(t-1)$ that mutually align for some $t \in [0,1]$. A final interpolation $\mathcal{I}(t)$ can then be computed by cross-dissolving both images:

$$\mathcal{I}(t) = (1-t) \cdot \mathcal{I}_1(t) + t \cdot \mathcal{I}_2(t-1)\,, \tag{4.1}$$

We implement blending again in a fragment shader, rendering the final interpolated image using efficient per pixel operations. Border regions, where only one image holds information, are selectively adopted. We also add a logistic sigmoid function from [Lev06] to regulate the blending during a transition from one image to another smoothly. It is controlled by the regularization parameter a, that can be interpreted as roughness or transition speed. For $a = 0$, the function collapses to the identity and Definition 4.1 becomes simple cross-dissolving. The logistic sigmoid becomes a step function for $a = 1$, that means no blending at all. Figure 4.3 shows some plots for different values of a. The



(a) $a = 0$. The logistic sigmoid collapses to the identity.

(b) $a = 0.787$. The transition increasingly concentrates around the center.

(c) $a = 0.987$. The sigmoid now resembles a step function with a very sharp transition.

Figure 4.3: Plots of the logistic sigmoid function for different values of the regularization parameter a.

sigmoid function is incorporated by using $sigmoid(t, a)$ instead of t in Definition 4.1. We choose $a = 0.5$, which suppresses early blending near source and target. This leads to sharper and more pleasing transitions, as blending is concentrated around the central part of a transition.

Definition 4.2: Logistic Sigmoid

The logistic sigmoid function is given by

$$\text{sigmoid}(x, a) = \frac{(1 + e^{-2\rho(x-0.5)})^{-1} - (1 + e^{\rho})^{-1}}{(1 + e^{-\rho})^{-1} - (1 + e^{\rho})^{-1}} \tag{4.2}$$

for $a \in [0, 1]$, where $\rho = (1 - \max(\epsilon, \min(1 - \epsilon, a)))^{-1} - 1$ and $\epsilon = 0.0001$.

The next and final section of this thesis presents applications, results and limitations of this work.

5 Results

In this chapter, we show some applications and results of the presented image interpolation framework. Additionally, we discuss the capabilities and limitations of our work. The thesis closes with a summary and outlook.

5.1 Applications

We tested and evaluated the proposed methods on several datasets, including the Middlebury Stereo and Flow datasets [BSL+11], images from the works of Seitz and Dyer [SD96], the Skater and Graffiti sequences introduced by Lipski *et al.* [LLBM09] and self captured scenes of which some could already be seen, namely Sony, Climbing, Kitchen, and KimWest.

The presented algorithms have been developed for temporal and spatial interpolation of multi-view video. Our initial goal was to provide a framework to achieve slow motion, virtual camera, and in combination Bullet Time effects. However, our framework can also be used for several other applications. These include photo tourism [SGSS08] and virtual avatars [Tim03]. For example, we implemented a virtual head, that follows a user in front of the monitor using face tracking. Instead of using some 3D model, we take a set of pictures showing a face from different angles. These pictures are then interpolated by means of the methods presented in this work. The following figures show some examples computed and rendered with our proposed image interpolation algorithms.

5.2 Discussion

For a more elaborate evaluation, we compare warped source images to their targets using a normalized sum of squared differences (NSSD) metric. Figure 5.7 shows averaged results from the presented datasets. We partition these results in separate classes, as sparse and dense techniques proof to be hardly comparable using the employed metric (inverse warping based approaches naturally yield better sampling quality (cf. Section 2.3)). It can clearly be seen in Figure 5.7, that the Dual TV-L1 optical flow outperforms its competitors. The Epipolar and ASIFT matching routines show a similar, average error rate. To incorporate evaluations between both classes, we additionally compare in-between

(a) Source (left) and target image (right). The static close-up section shown below is marked.

(b) Uniform transition from $t = 0$ (left, source image) to $t = 1$ (right, target image) using the Epipolar Guided Matcher, sparse warping and image blending. For clarity, only the middle section is shown in detail.

Figure 5.1: Room scene. The figure shows a static close-up of a smooth transition from source to target image. Our framework can be applied to any sequence of temporally or spatially related images. One can imagine numerous other useful examples, like the exploration of buildings based on snapshots taken along the way.

(a) *From left to right*: Source image, target image. Again, the close-up section is marked.

(b) Details of interpolations at $t = 0.5$ using different algorithms. *From left to right*: Simple cross-dissolve, porposed interpolation using the Epiolar Guided Matcher, Dual TV-L1 based interpolation.

Figure 5.2: Our approach yields plausible transformations between images, even for zooming and rotating camera motions. Notice the huge difference between naive image blending (left) and our approach (middle).

Figure 5.3: Avatar scene. The graphic shows interpolations at $t = 0.5$. *From left to right*: Source image, simple cross-dissolve, Dual TV-L1 based interpolation, target image. Without warping, obvious ghosting artifacts are introduced. Our framework on the other hand produces smooth and plausible transitions.

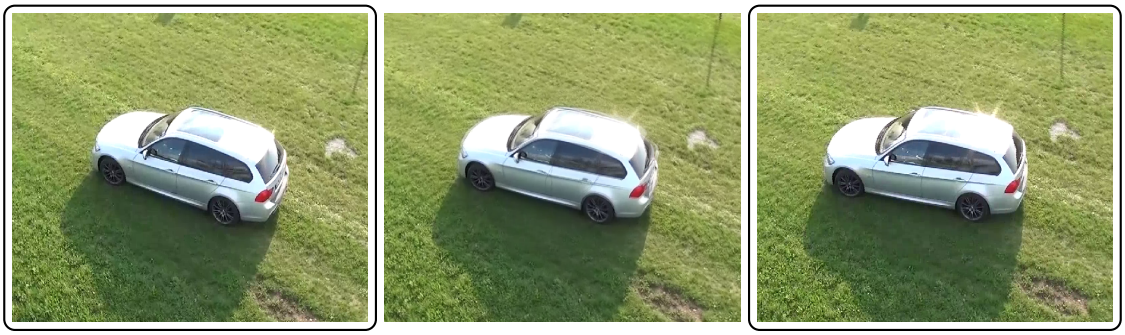
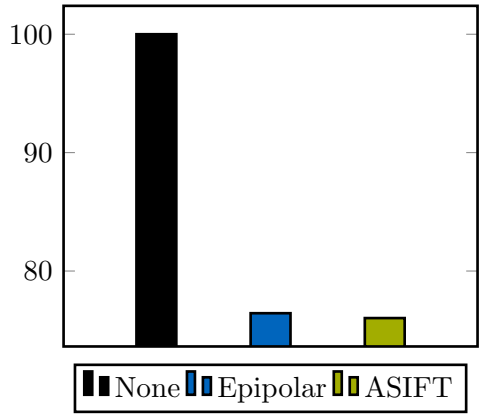


Figure 5.4: BMW scene at 40 skipped frames. *From left to right*: Source image, ASIFT-based interpolation at $t = 0.5$, target image. Our framework yields plausible in-between images. Only few ghosting artifacts are visible in the interpolation.

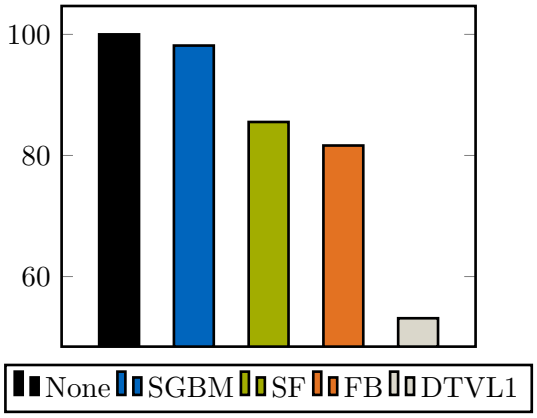
Figure 5.5: Graffiti scene [LLBM09]. *From left to right*: Source image, mesh warping based interpolation at $t = 0.5$, target image. While the foreground suffers from distortions and ghosting, the overall scene motion is well represented.

(a) BMW scene (b) Graffiti scene

Figure 5.6: For comparison, this figure shows the previous interpolation examples using naive blending without warping. Notice the severe ghosting artifacts.



(a) Sparse warping methods. The Epipolar and ASIFT matcher show similar, average error rates. Further evaluation is done based on the interpolated images themselves.



(b) Dense warping methods. The Dual TV-L1 (DTVL1) optical flow clearly outperforms its competitors Semi-Global Block Matching (SGBM), SimpleFlow (SF) and Farneback (FB).

Figure 5.7: Comparison of different warping techniques. *X-axis*: Algorithm. *Y-axis*: Average *normalized sum of squared differences* between target and destination image (*percent*). The values are given relative to the distance between the original source and target images.

images at $t = 0.5$ to ground truth data (original, skipped frames) for different algorithms and baselines. As seen in the previous chapters, dense approaches work well for temporal and small spatial image variations, the Epipolar matcher works well on small to medium baselines and ASIFT proves of use for extreme changes of view, like point of interest shots. The output quality for all algorithms also depends on the input image size. By downsampling input sequences in a pre-processing step, the density and accuracy of matchings can be influenced. For most examples, we use images that were downsampled to 480 px in height, mostly to save computation time, allowing for quick evaluation.

The initial goal of this thesis was to provide an interpolation pipeline, that is able to automatically generate 360° views of arbitrary, dynamic, aerial captured scenes. Our proposed framework yields good results for a wide range of input sequences, reaching from indoor video captures to spatially varying outdoor scenes. However, the presented algorithms are limited to certain criteria. For optical flow based methods, we found that input videos should have a minimum framerate of 30 fps. Even then, highly dynamic scene contents often moves too fast to be accurately captured by optical flow algorithms. This leads to artifacts and significant errors in the output images. Therefore an input framerate of 60 fps or above is recommended. Spatial interpolation using feature-based matching and mesh warping on the other hand is highly dependend on scene motion. It yields very good results for constant background colors and when the scene content

Figure 5.8: Skater scene from Lipski *et al.* [LLBM09]. The graphic shows a transition from source (left) to target image (right), with $t = 0.5$ at the middle panel. Distortions and blending artifacts occur due to the large relative background motion. This is a general limitation of mesh-warping based approaches. It could for example be solved using background/foreground segmentation.

underlies certain affine transformations, namely camera translation or panorama shots. It fails however when the relative motion between foreground and background increases, e.g. for significant latitudal or longitudal camera rotations and background parallalax. This is a general limitation of mesh warping based approaches, as they lack the ability to deal with depth (i.e. occlusions). It can for example be observed comparing our approach to that of Lipski *et al.* on the Skater dataset [LLBM09]. As Figure 5.8 shows, the large relative background motion leads to distortions in the foreground, as its triangles deform the whole region. Different, more sophisticated approaches exist to deal with occlusions and more complex scenes, e.g. image segmentation, depth based rendering, billboards or graph cuts, as explored in numerous papers (cf. [ZKU$^+$04, LLM10, LLBM09, IS05, BBPP10, KL10]).

Besides, in the final version of our framework, we offer one dimensional interpolation only, that is either temporal or spatial interpolation can be applied. This is by design as the combination of two warping steps introduces even more artifacts as already might occur. Two dimensional interpolation can be achieved by considering not pairs but four adjacent images and applying bilinear interpolation, first interpolating two image pairs along one axis and then interpolating the resulting images along the other. Additionally, the displacement maps have to be interpolated. This is no trivial task, especially not for pairwise sparse matchings. One would have to manually interpolate attributed triangulations, which is currently done by the graphics hardware and OpenGL. Nevertheless,

we implemented and explored this method for the Avatar2D scene, which can be found in the supplemental material.

5.3 Outlook

We presented algorithms for both temporal and spatial image interpolation. They consist of image matching, warping and blending. We showed how dense image correspondences, established using optical flow methods, can be used to create in-between images of image pairs with minor variations, e.g. temporal image sequences. Then, different feature detectors, descriptors and sparse matching approaches were covered, including sophisticated outlier elimination methods. After outlining delaunay triangulation and match extrapolation, we presented mesh warping using sparse feature correspondences. Additionally, the extrapolation of matches along image borders and the idea of pre- and postwarping to achieve physically valid in-between renderings were covered.

We are quite happy with the results. Unfortunately, the proposed image interpolation pipeline is more limited than we hoped it to be, but this had been expected. Further work can be done, building upon this thesis, and we look forward to employ more sophisticated matching, warping and blending alternatives. We especially look forward to incorporate edge information, image segmentation and occlusion handling, as done by Zitnick *et al.* [ZKU+04] and Lipski *et al.* [LLBM09].

List of Figures

List of Tables

List of Definitions

List of Algorithms

Bibliography

[AS97] Shai Avidan and Amnon Shashua. Novel view synthesis in tensor space. In *CVPR*, pages 1034–1040. IEEE Computer Society, 1997.

[BBPP10] Luca Ballan, Gabriel J. Brostow, Jens Puwein, and Marc Pollefeys. Unstructured video-based rendering: interactive exploration of casually captured videos. *ACM Trans. Graph.*, 29(4), 2010.

[BETG08] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer Vision and Image Understanding (CVIU)*, 110:346–359, 2008.

[BN92] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In James J. Thomas, editor, *SIGGRAPH*, pages 35–42. ACM, 1992.

[Bow81] A. Bowyer. Computing dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981.

[BSL+11] Simon Baker, Daniel Scharstein, J.P. Lewis, Stefan Roth, MichaelJ. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

[CL12] Kanglin Chen and Dirk A. Lorenz. Image sequence interpolation based on optical flow, segmentation, and optimal control. *IEEE Transactions on Image Processing*, 21(3):1020–1030, 2012.

[CW93] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *SIGGRAPH*, pages 279–288. ACM, 1993.

[Far03] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In Josef Bigün and Tomas Gustavsson, editors, *SCIA*, volume 2749 of *Lecture Notes in Computer Science*, pages 363–370. Springer, 2003.

[FB81] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[FFR+12] Pasqualina Fragneto, Andrea Fusiello, Beatrice Rossi, Luca Magri, and Matteo Ruffini. Uncalibrated view synthesis with homography interpolation. In *3DIMPVT*, pages 270–277. IEEE, 2012.

[Fun00] Thomas Funkhouser. Cs426: Image warping, 2000.

[Har99] Richard I. Hartley. Theory and practice of projective rectification. *International Journal of Computer Vision*, 35(2):115–127, 1999.

[Hir08] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE TPAMI*, 30(2):328–341, 2008.

[HS81] Berthold K P Horn and Brian G Schunk. Determining Optical Flow. *Artificial Intelligence*, 17:185–203, 1981.

[HS88] C Harris and M Stephens. A Combined Corner and Edge Detection. In *Proceedings of The Fourth Alvey Vision Conf.*, pages 147–151, 1988.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[Inc13] Replay Technologies Inc. freed™ technology, 2013.

[IS05] Naho Inamoto and Hideo Saito. Free viewpoint video synthesis and presentation from multiple sporting videos. In *ICME*, pages 322–325. IEEE, 2005.

[Jac91] Michael Jackson, November 1991.

[KL10] Shanat Kolhatkar and Robert Laganière. Real-time virtual viewpoint generation on the gpu for scene navigation. In *CRV*, pages 55–62. IEEE, 2010.

[Lag11] Robert Laganière. *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing, May 2011.

[Lev06] Golan Levin. Exponential shaping functions, 2006.

[Lip13] Christian Lipski. *Virtual Video Camera: a System for Free Viewpoint Video of Arbitrary Dynamic Scenes*. PhD thesis, TU Braunschweig, June 2013.

[LLBM09] Christian Lipski, Christian Linz, Kai Berger, and Marcus A. Magnor. Virtual video camera: image-based viewpoint navigation through space and time. In *SIGGRAPH Posters*. ACM, 2009.

[LLM10] Christian Linz, Christian Lipski, and Marcus A. Magnor. Multi-image interpolation based on graph-cuts and symmetric optical flow. In *SIGGRAPH Posters*. ACM, 2010.

[Low99] David G. Lowe. Object recognition from local scale-invariant features. In *Proc. of the International Conference on Computer Vision, Corfu*, 1999.

[Low03] David G. Lowe. Distinctive image features from scale-invariant keypoints. In *International Journal of Computer Vision*, volume 20, 2003.

[MD99] Russell A. Manning and Charles R. Dyer. Interpolating view and scene

motion by dynamic view morphing. In *CVPR*, pages 1388–1394. IEEE Computer Society, 1999.

[ML09] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In Alpesh Ranchordas and Helder Araújo, editors, *VISAPP (1)*, pages 331–340. INSTICC Press, 2009.

[MY09] Jean-Michel Morel and Guoshen Yu. Asift: A new framework for fully affine invariant image comparison. *SIAM J. Imaging Sciences*, 2(2):438–469, 2009.

[Røs08] Eivind Røssaak. *Negotiating Immobility*. Acta Humaniora. Unipub AS, 2008.

[SD95] Steven M. Seitz and Charles R. Dyer. Physically-valid view synthesis by image interpolation. In *In Proc. IEEE Workshop on Representations of Visual Scenes*, pages 18–25, 1995.

[SD96] Steven M. Seitz and Charles R. Dyer. View morphing. In *SIGGRAPH*, pages 21–30, 1996.

[SGSS08] Noah Snavely, Rahul Garg, Steven M. Seitz, and Richard Szeliski. Finding paths through the world's photos. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3):11–21, 2008.

[SPMLF13] Javier Sánchez Pérez, Enric Meinhardt-Llopis, and Gabriele Facciolo. TV-L1 Optical Flow Estimation. *Image Processing On Line*, 2013:137–150, 2013.

[Sze11] Richard Szeliski. *Computer vision algorithms and applications*. Springer, London; New York, 2011.

[TBKP12] Michael W. Tao, Jiamin Bai, Pushmeet Kohli, and Sylvain Paris. Simpleflow: A non-iterative, sublinear optical flow algorithm. *Computer Graphics Forum (Eurographics 2012)*, 31(2), May 2012.

[Tim03] Karl Timm. Real-time view morphing of video streams. In Alyn P. Rockwood, editor, *SIGGRAPH*. ACM, 2003.

[TM07] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: A survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2007.

[TS94] C. Tomasi and J. Shi. Good features to track. In *Proc. IEEE Conf. on Comp. Vision and Patt. Recog.*, pages 593–600, 1994.

[Vla10] A. Vlad. Image morphing techniques. *JIDEG*, 5(1), JUNE 2010.

[Wol98] G. Wolberg. Image morphing : A survey. *Visual Computer*, 14:360–372, 1998.

[WPUB11] Manuel Werlberger, Thomas Pock, Markus Unger, and Horst Bischof. Optical flow guided tv-l1 video interpolation and restoration. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, 2011.

[XS04] Jiangjian Xiao and Mubarak Shah. Tri-view morphing. *Computer Vision and Image Understanding*, 96(3):345–366, 2004.

[ZDFL95] Zhengyou Zhang, Rachid Deriche, Olivier D. Faugeras, and Quang-Tuan Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artif. Intell.*, 78(1-2):87–119, 1995.

[ZKU+04] C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon A. J. Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, 23(3):600–608, 2004.